Low Complexity Link-state Multi-path Routing

Pascal Mérindol

Université catholique de Louvain
Louvain-la-Neuve, Belgium

Jean Jacques Pansiot
Stéphane Cateloin

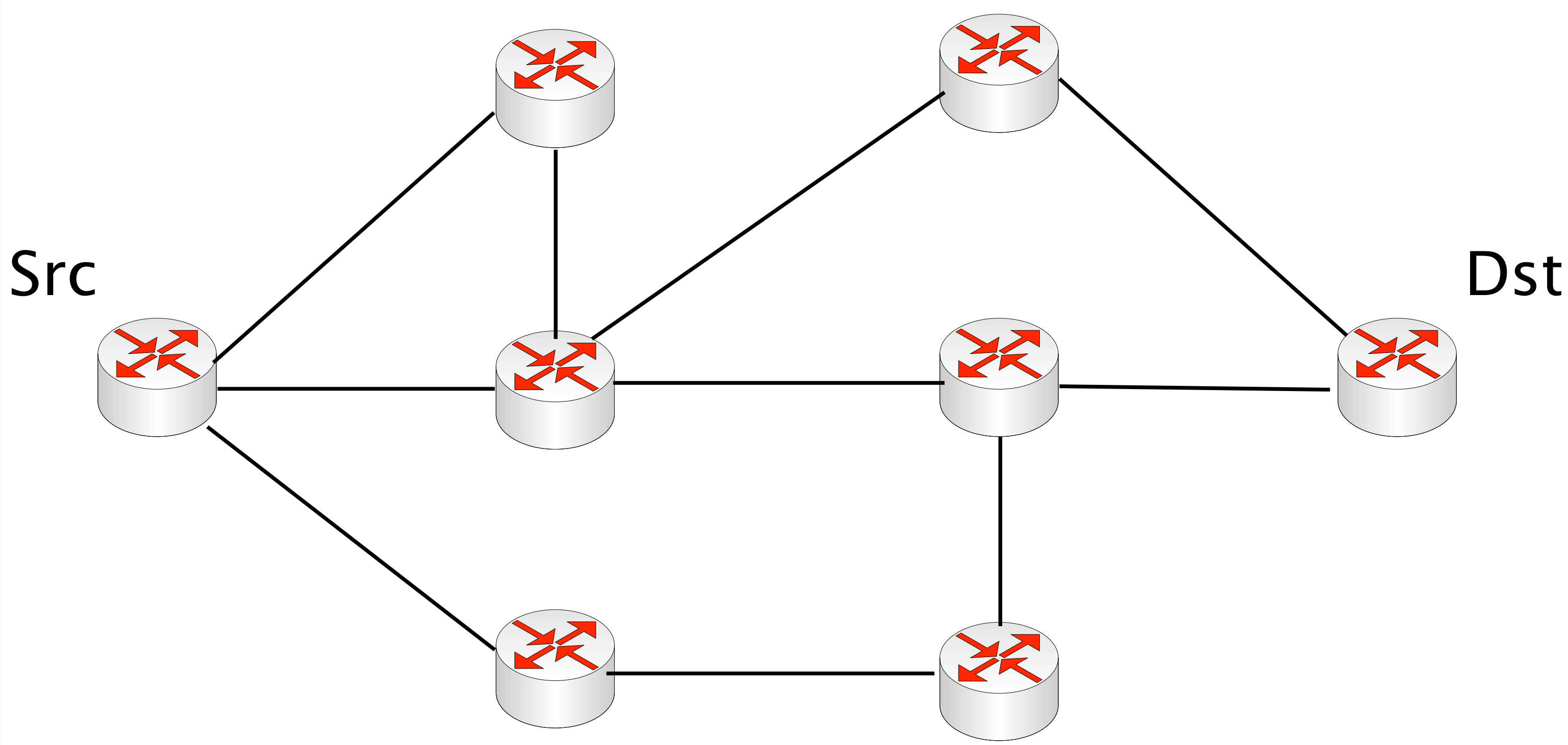
Université de Strasbourg
Strasbourg, France

Global Internet 2009
Infocom 2009 workshop
Rio de Janeiro, Brazil
24 April 2009

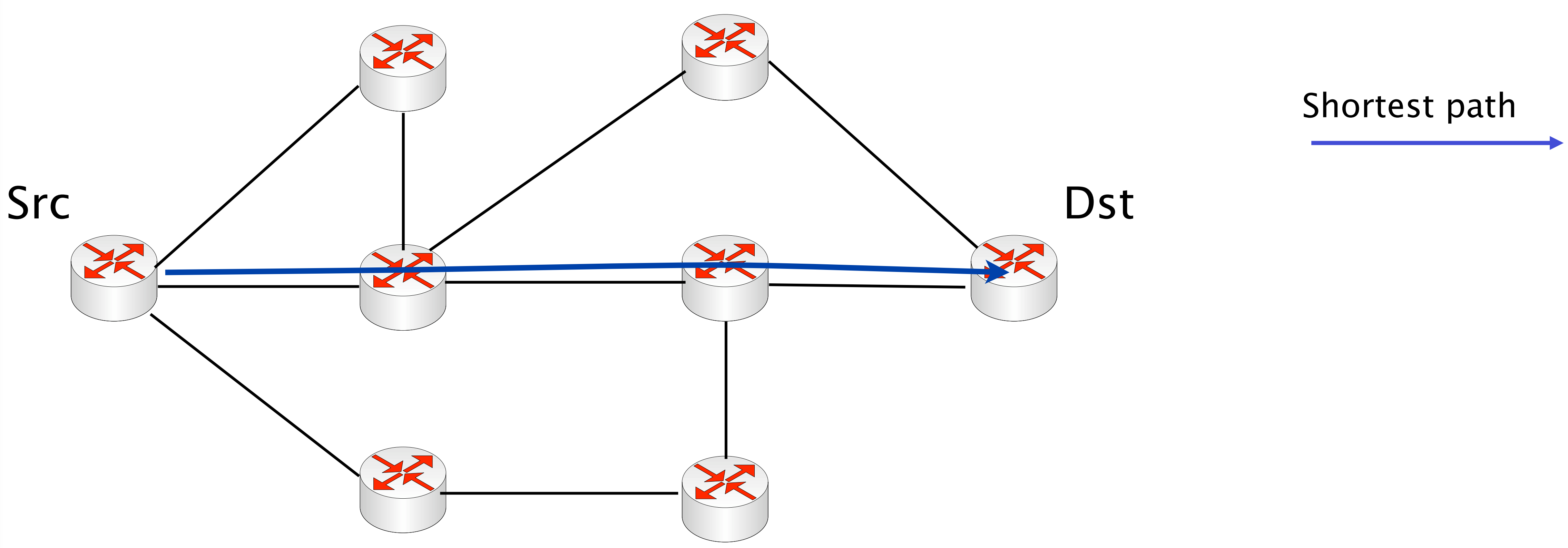
Talk outlines

1. Context and multi-path routing motivations
2. Graph and transverse paths terminology
3. Transverse paths properties
4. The *multi-Dijkstra-Transverse* (***mDT***) algorithm
5. Evaluation results

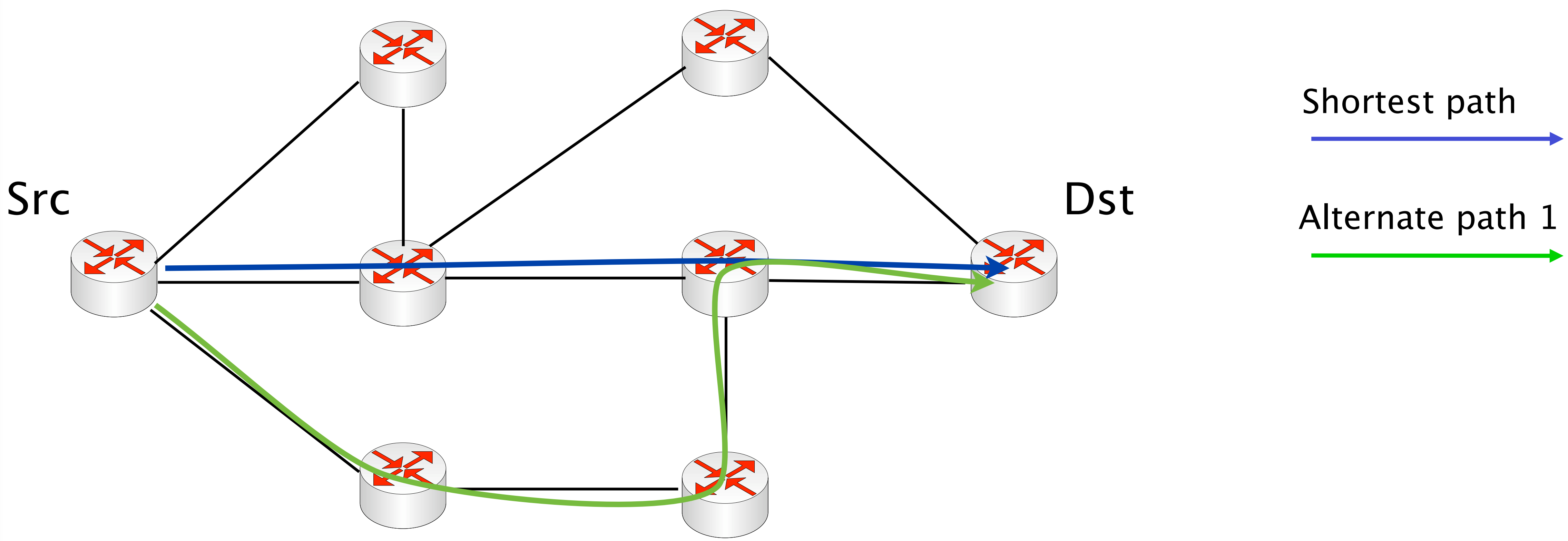
Multi-path routing objectives



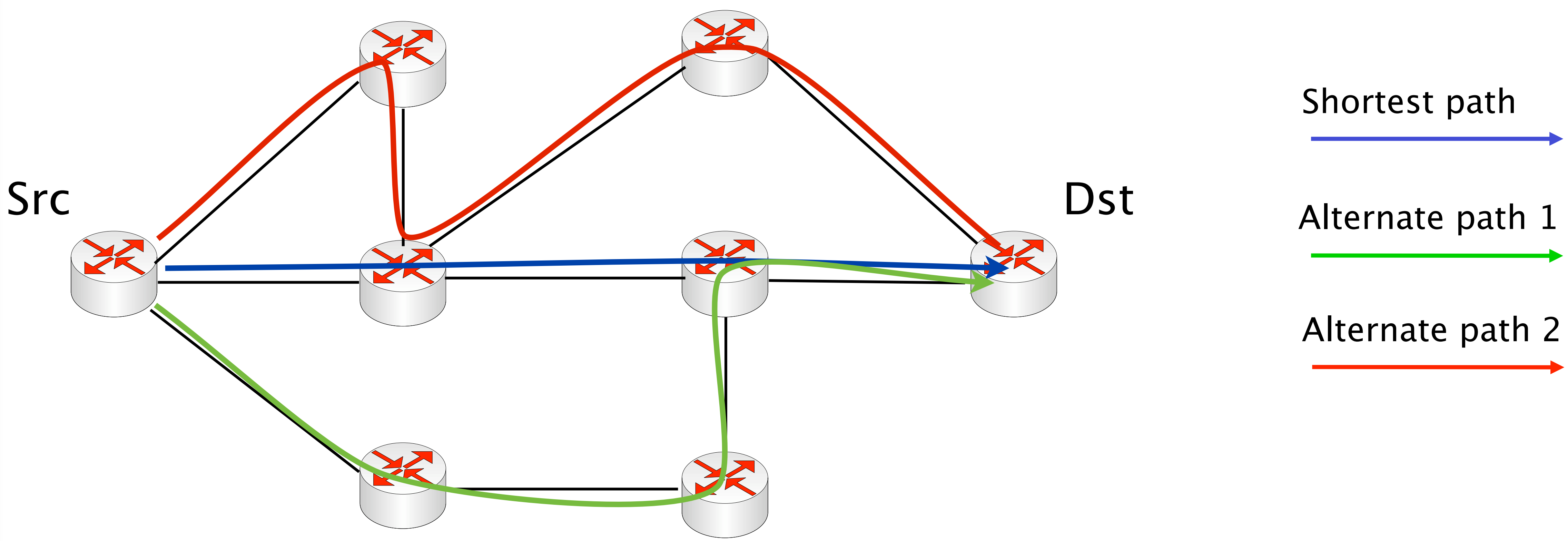
Multi-path routing objectives



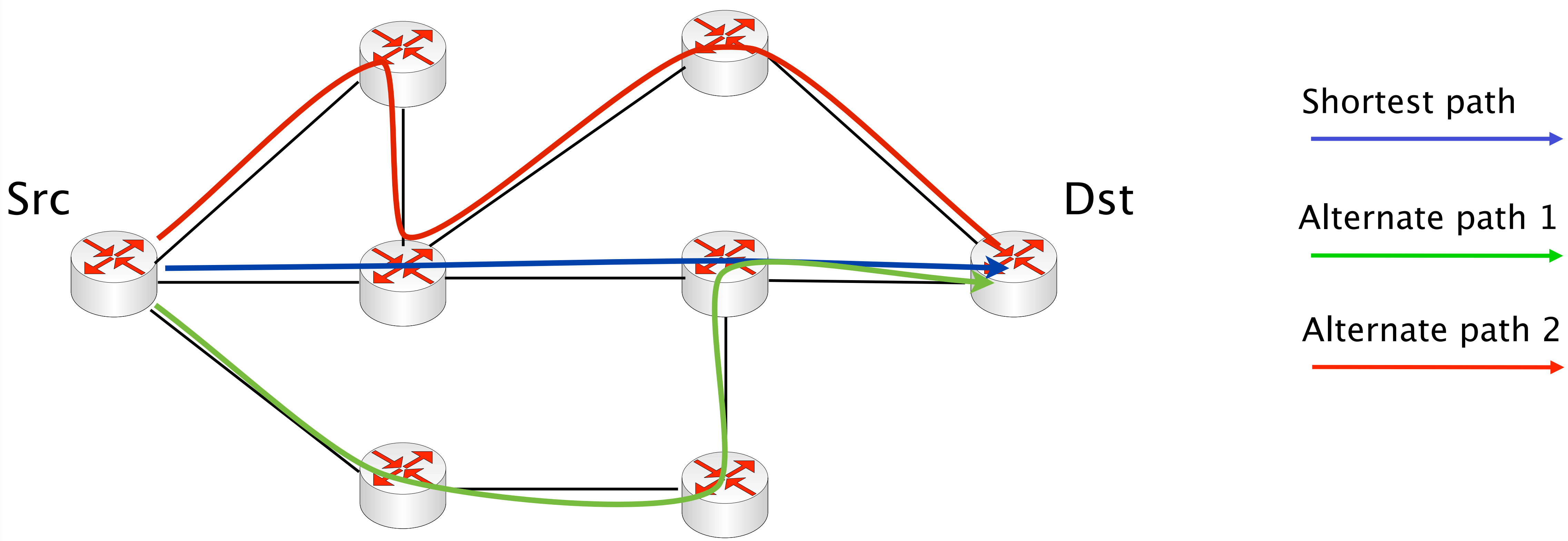
Multi-path routing objectives



Multi-path routing objectives

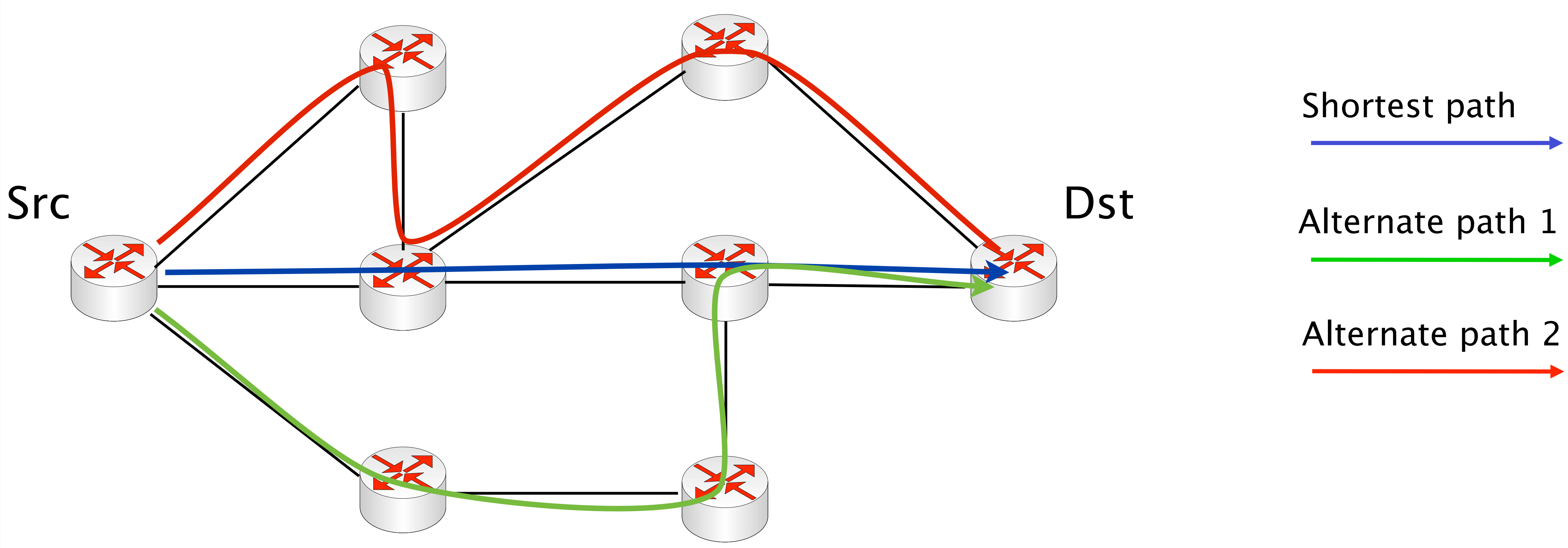


Multi-path routing objectives



- Goals of the path diversity:

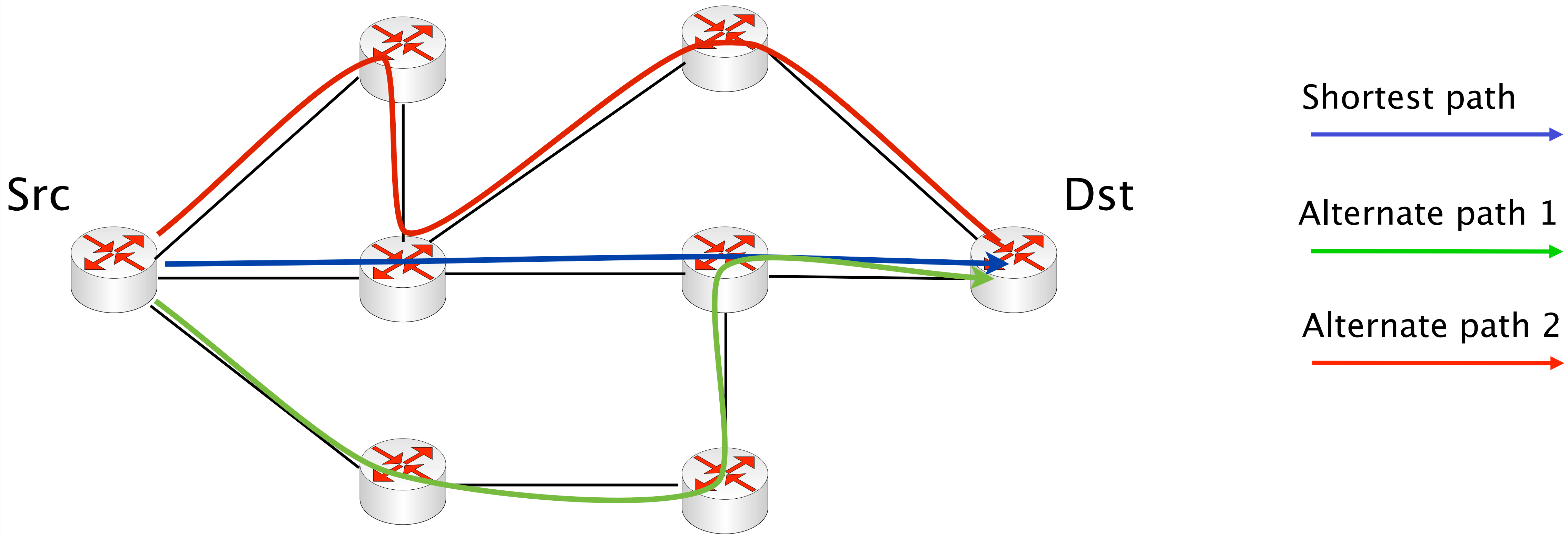
Multi-path routing objectives



- Goals of the path diversity:

1. Reliability : restoration time decreases

Multi-path routing objectives



- Goals of the path diversity:

1. Reliability : restoration time decreases
2. Load balancing (in the core or at the edges) : throughput increases, latency decreases

Context

- Intra-domain multi-path routing
- Link state protocol (e.g, OSPF or IS-IS):

1. Link state advertisements flooding
2. Additive metric (e.g, the sum of the inverse of the link capacities or the number of hops...)
3. Hop by hop forwarding

- For each calculating node, the root of the Shortest Path Tree (SPT), the goal is to validate alternate next hops guaranteeing the correctness of the forwarding
  (the distributed composition of next hops along the route does not induce rooting loops)

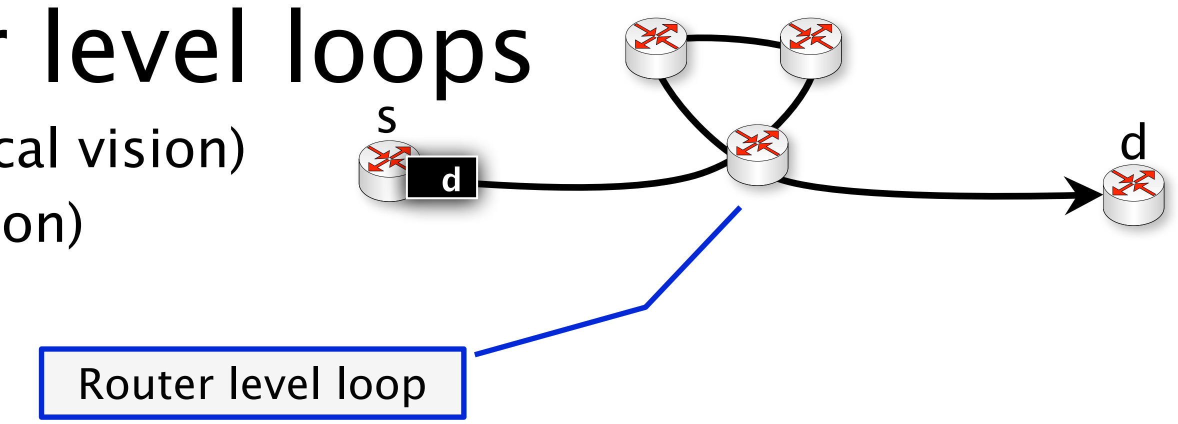
Goals and constraints

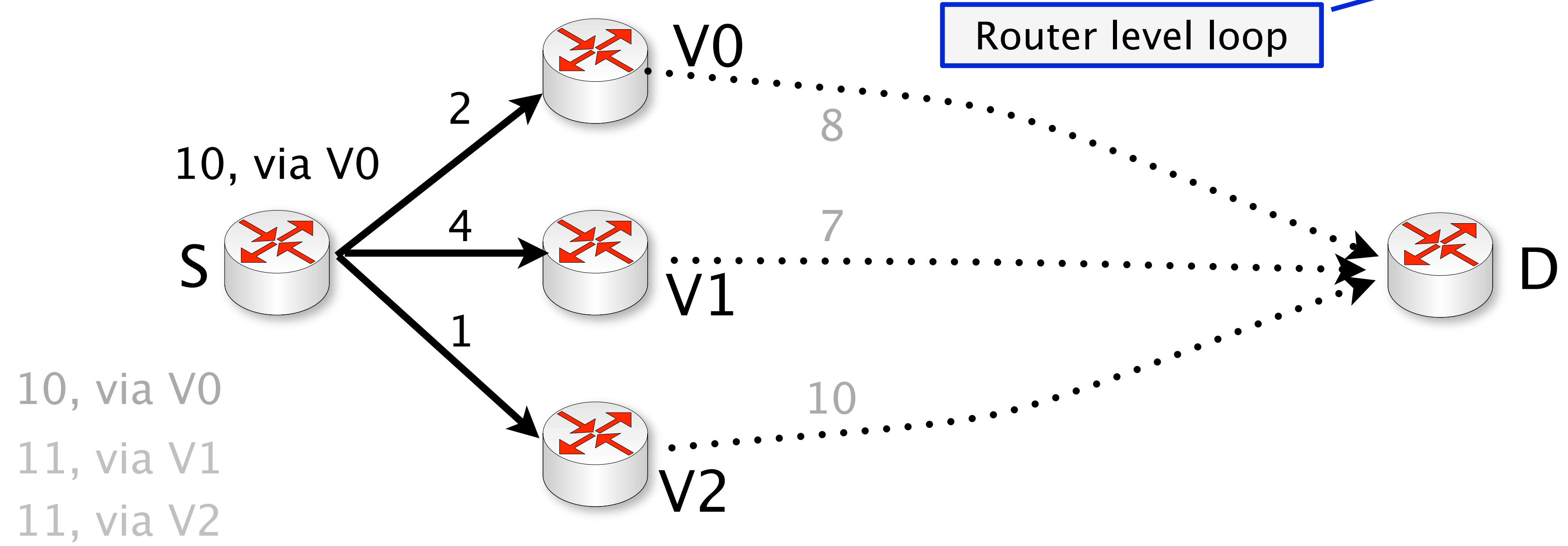
- We want:
  - Load balancing (for traffic engineering)
  - Reliability (for fast reroute)

- ...but we also want to ensure:
  - Loop free routing (correctness property)
  - Incremental deployment (without message exchange)
  - Low time and memory complexity overhead compared to Dijkstra
    - ➡Avoid the computation of the SPT of each neighbor router (kD): the time complexity of this approach is proportional to the calculating router degree, k

Simplest loop-free rules

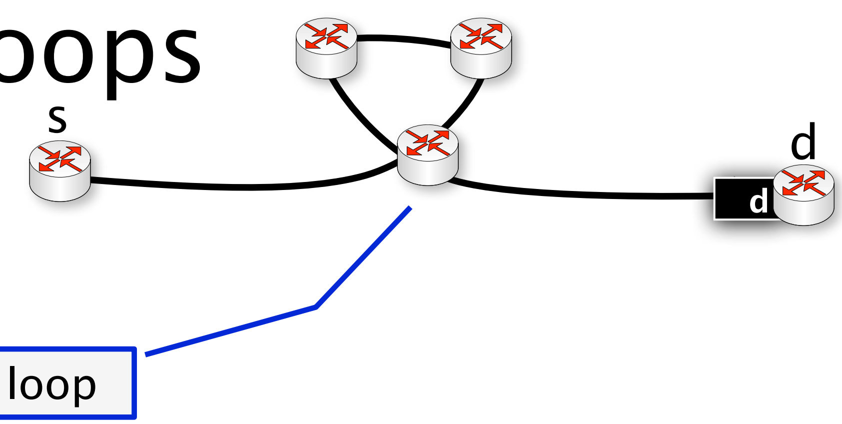
- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)

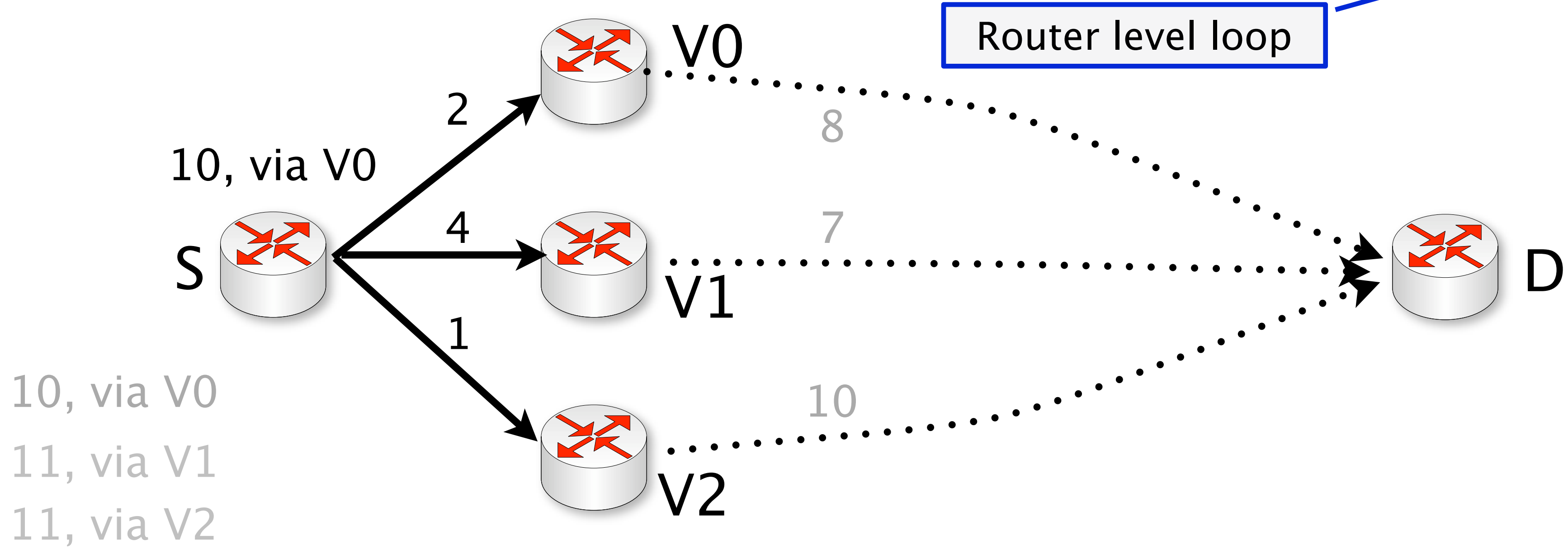




Simplest loop-free rules

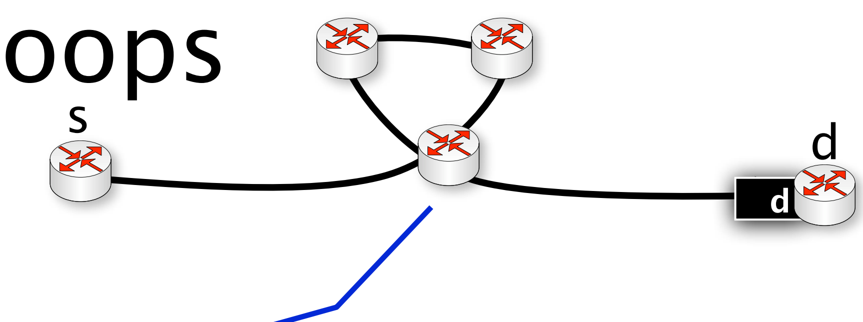
- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)

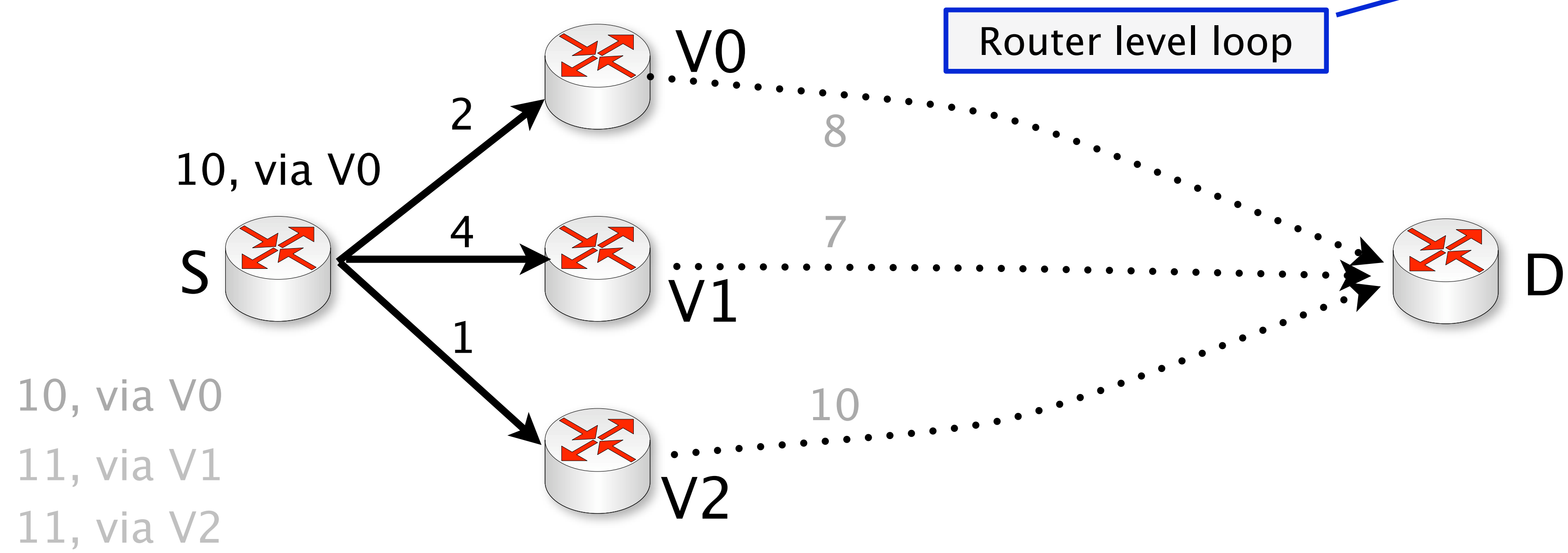




Simplest loop-free rules

- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)

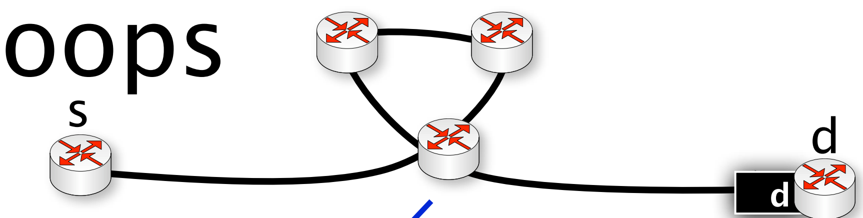


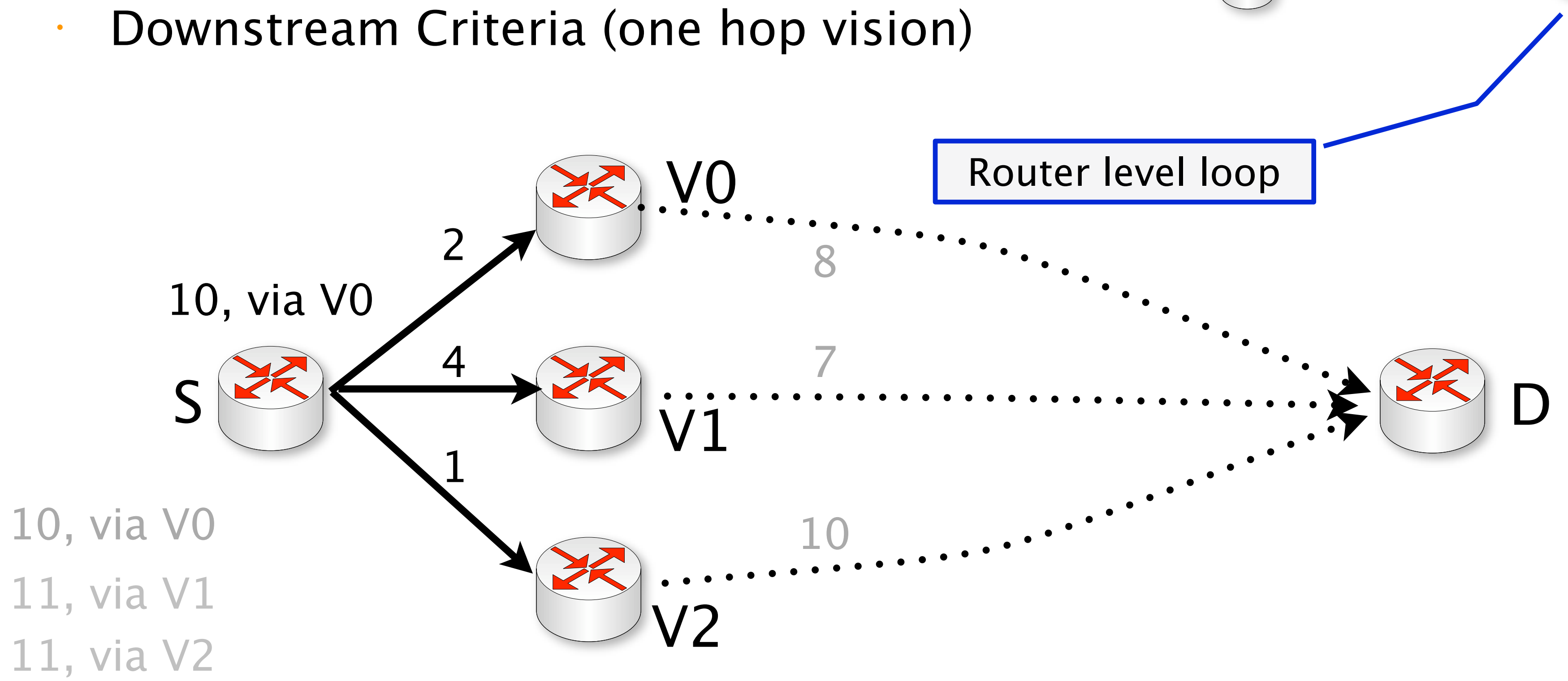


- How to know the costs of the neighbors ?

Simplest loop-free rules

- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)



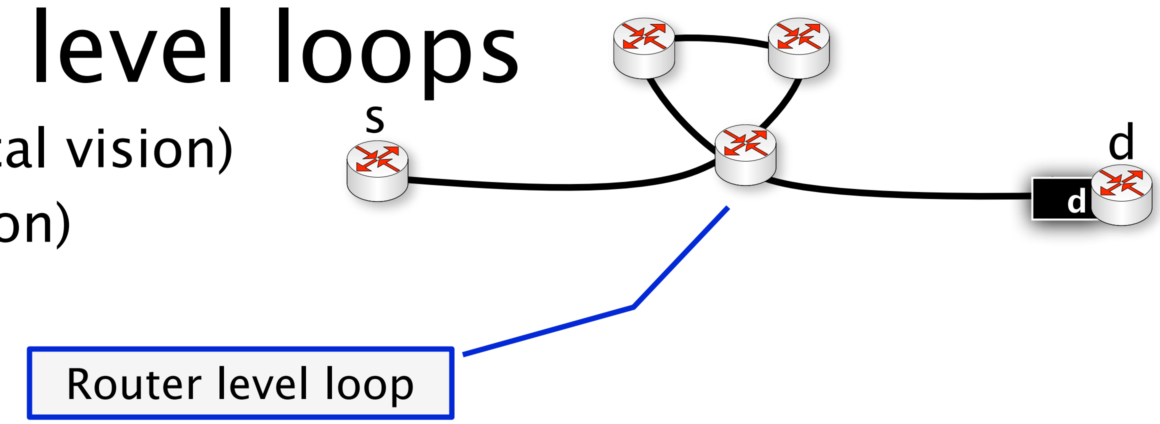


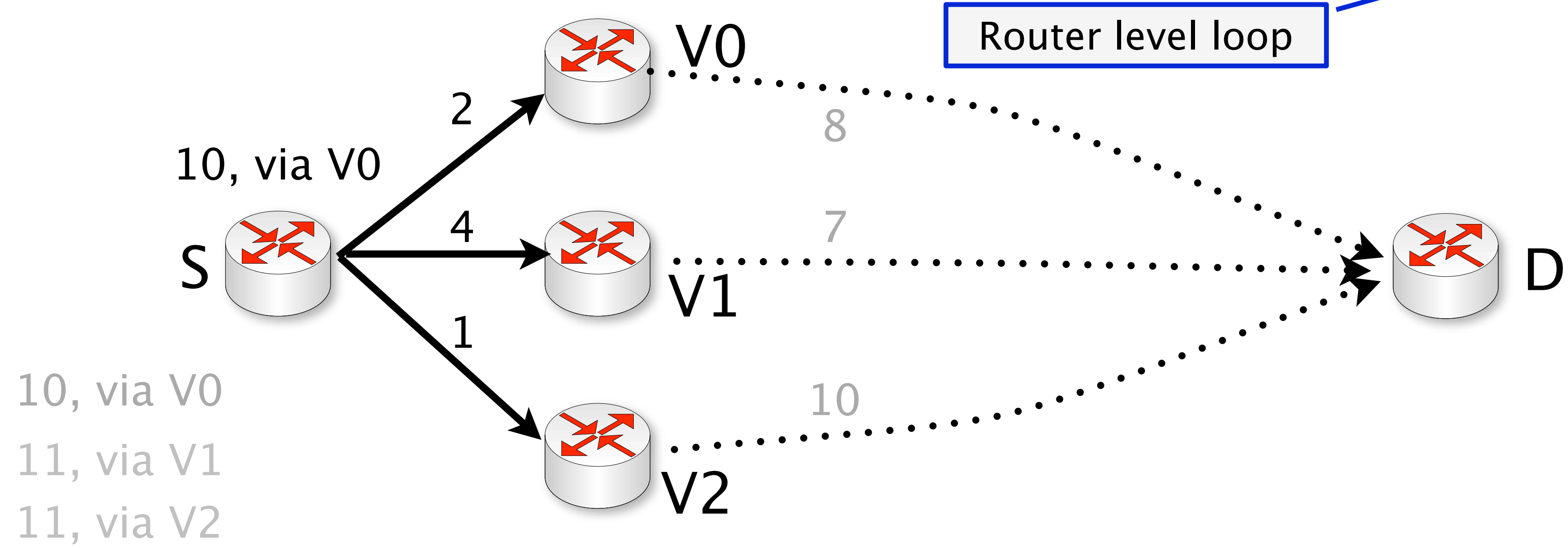
- How to know the costs of the neighbors ?

1.Send cost queries to the neighborhood (~ distance vector protocol) ?

Simplest loop-free rules

- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)



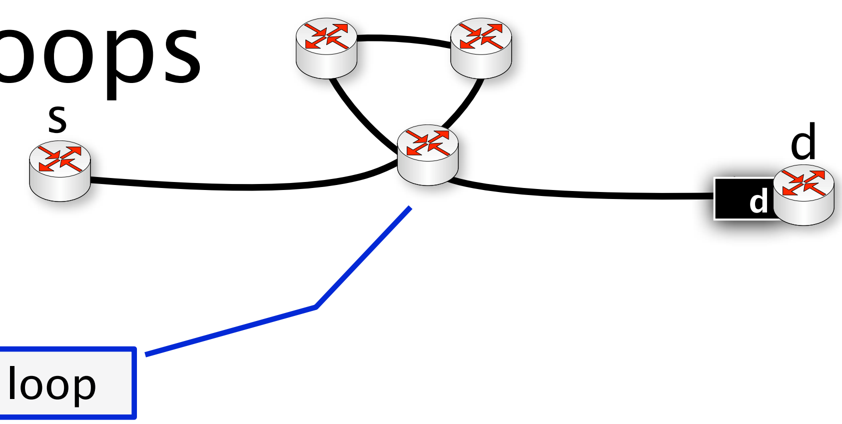


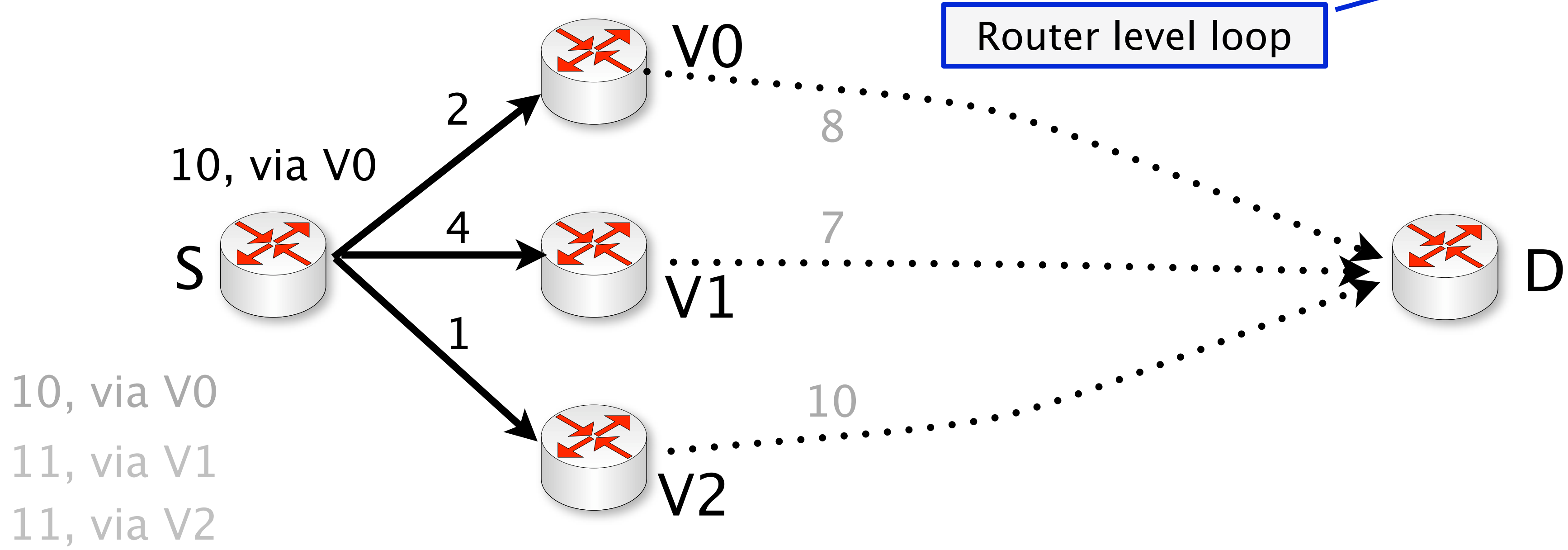
- How to know the costs of the neighbors ?

1. Send cost queries to the neighborhood (~ distance vector protocol) ?
2. Compute the SPT of each neighbor (kD) ?

Simplest loop-free rules

- Rules to avoid router level loops
  - Equal Cost Multi-path Routing (local vision)
  - Downstream Criteria (one hop vision)





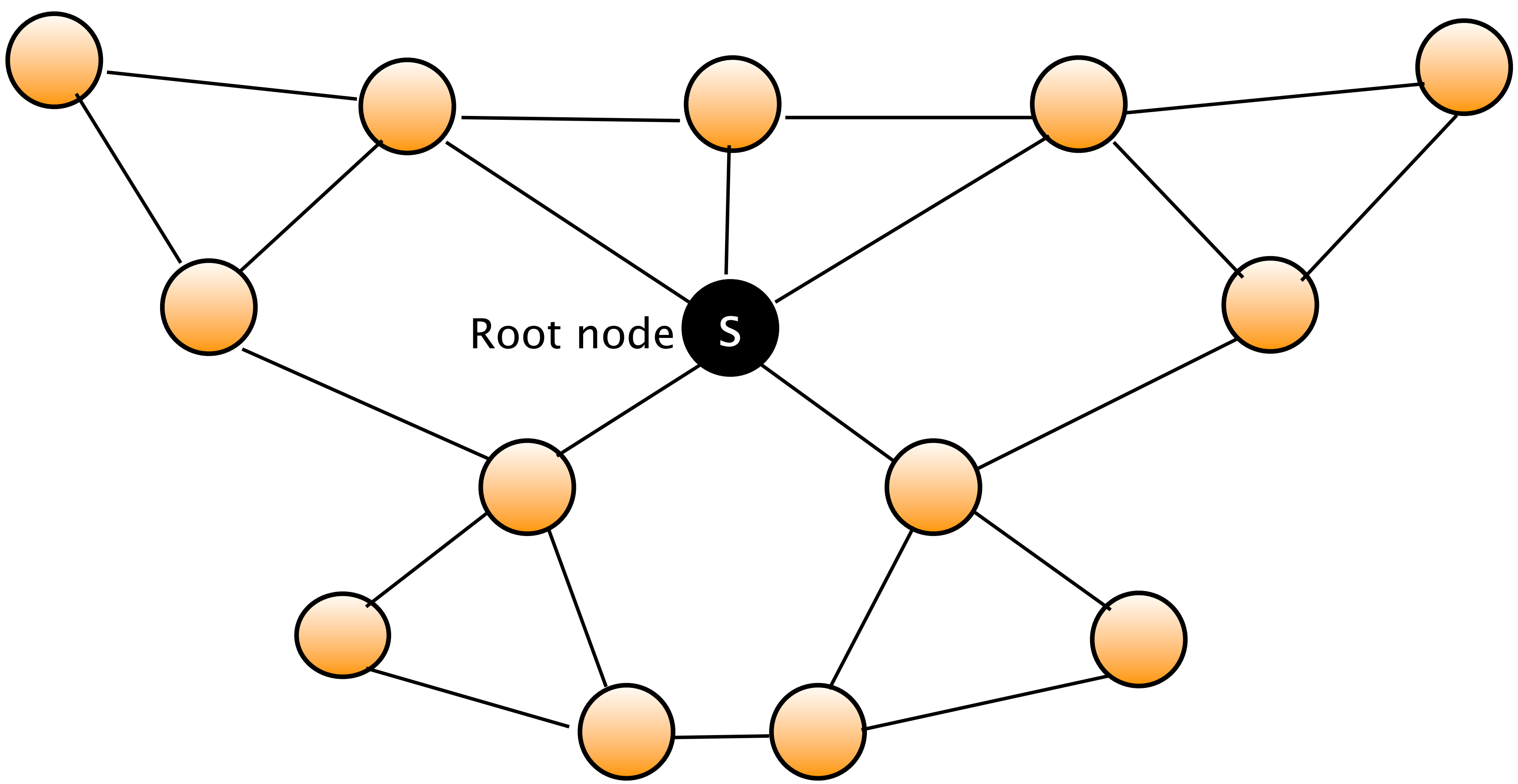
- How to know the costs of the neighbors ?

1. Send cost queries to the neighborhood (~ distance vector protocol) ?
2. Compute the SPT of each neighbor (kD) ?
3. Compute *candidate* paths with distinct outgoing interfaces thanks to an enhanced shortest path first (SPF) algorithm ?

Graph terminology

- The edges of a graph can be partitioned into 3 categories (considering both directions):

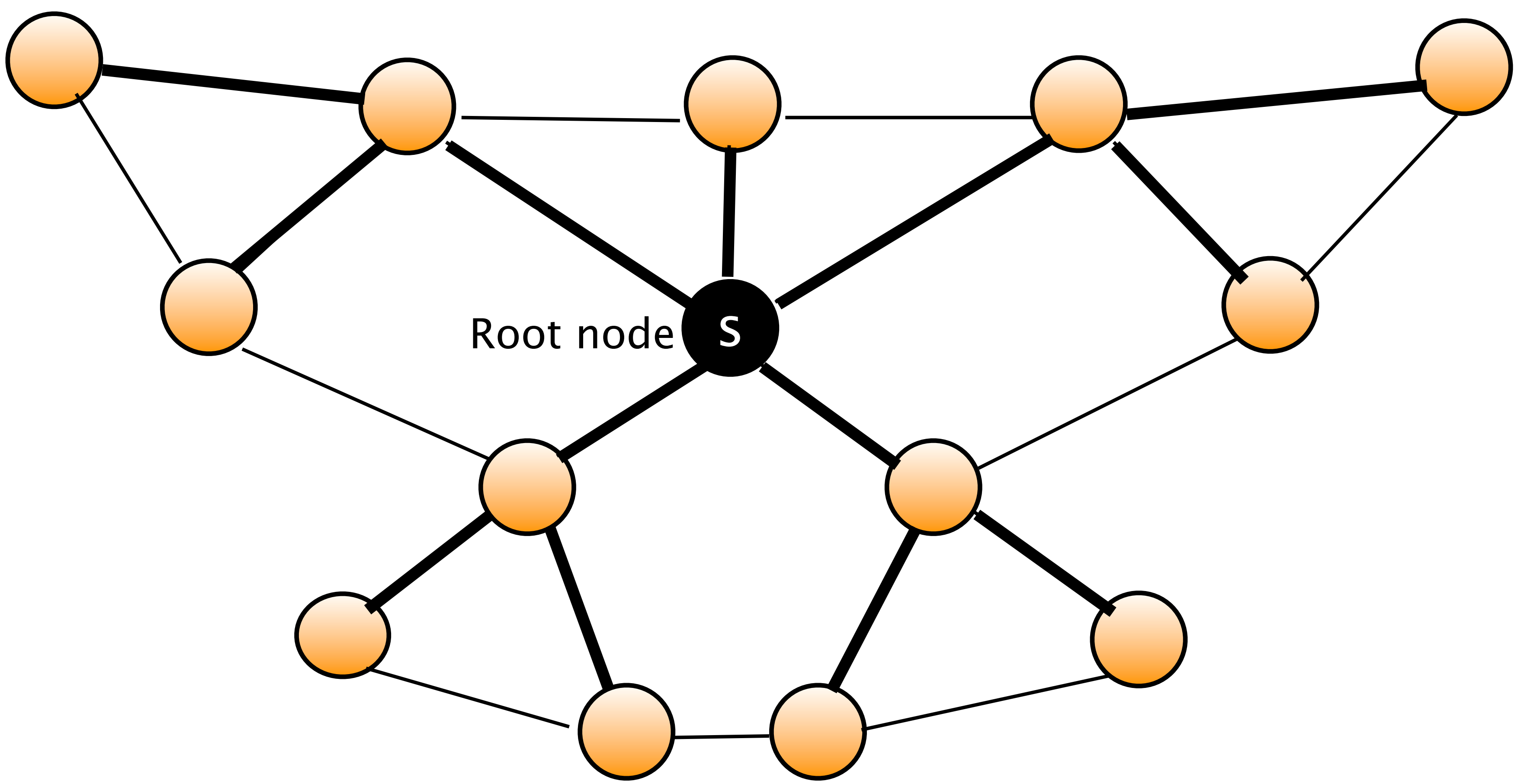
1. SPT edges: links belonging to the SPT
2. Transverses edges: links between branches
3. Internal edges : links between nodes of the same branch



Graph terminology

- The edges of a graph can be partitioned into 3 categories (considering both directions):

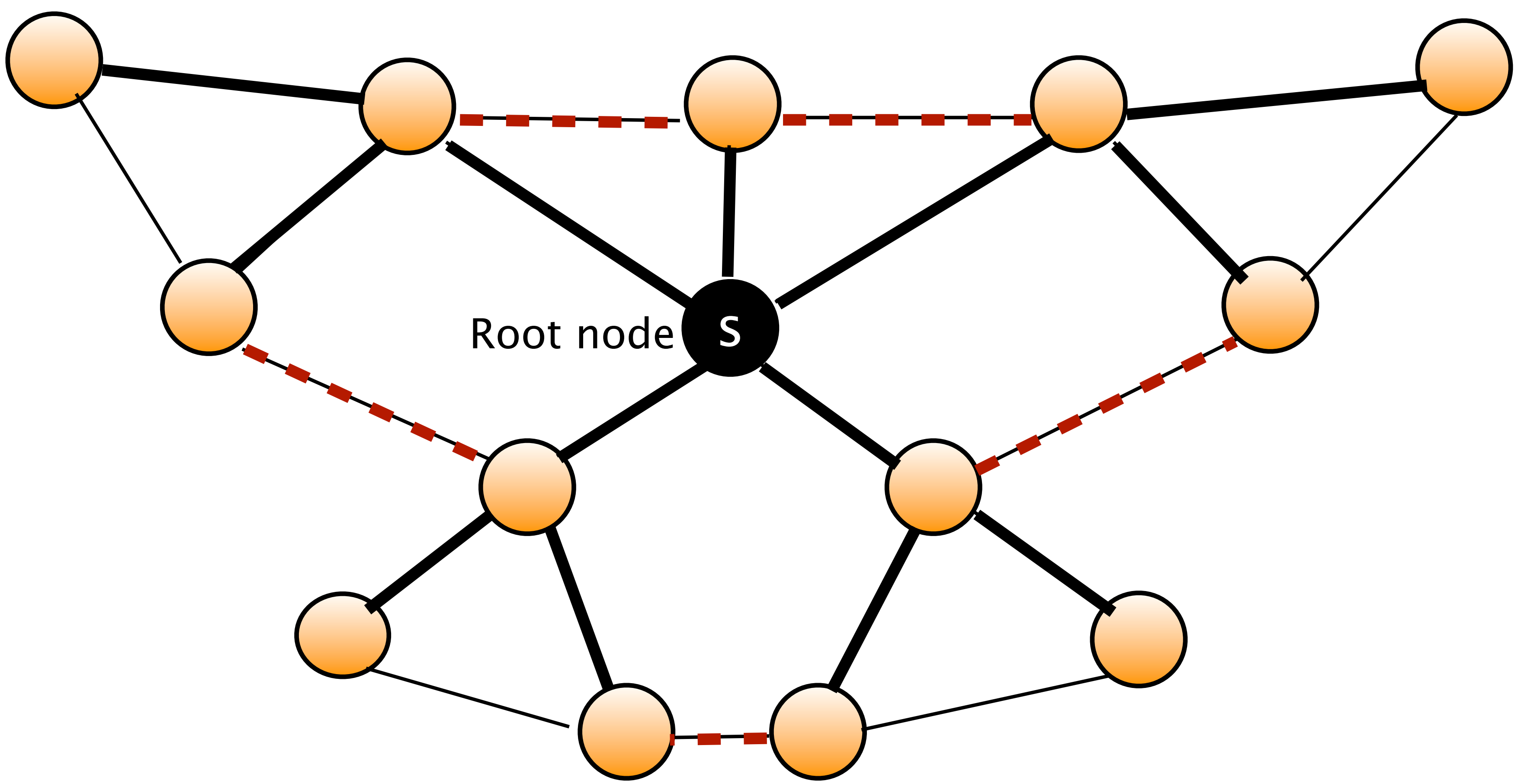
1. SPT edges: links belonging to the SPT
2. Transverses edges: links between branches
3. Internal edges : links between nodes of the same branch



Graph terminology

- The edges of a graph can be partitioned into 3 categories (considering both directions):

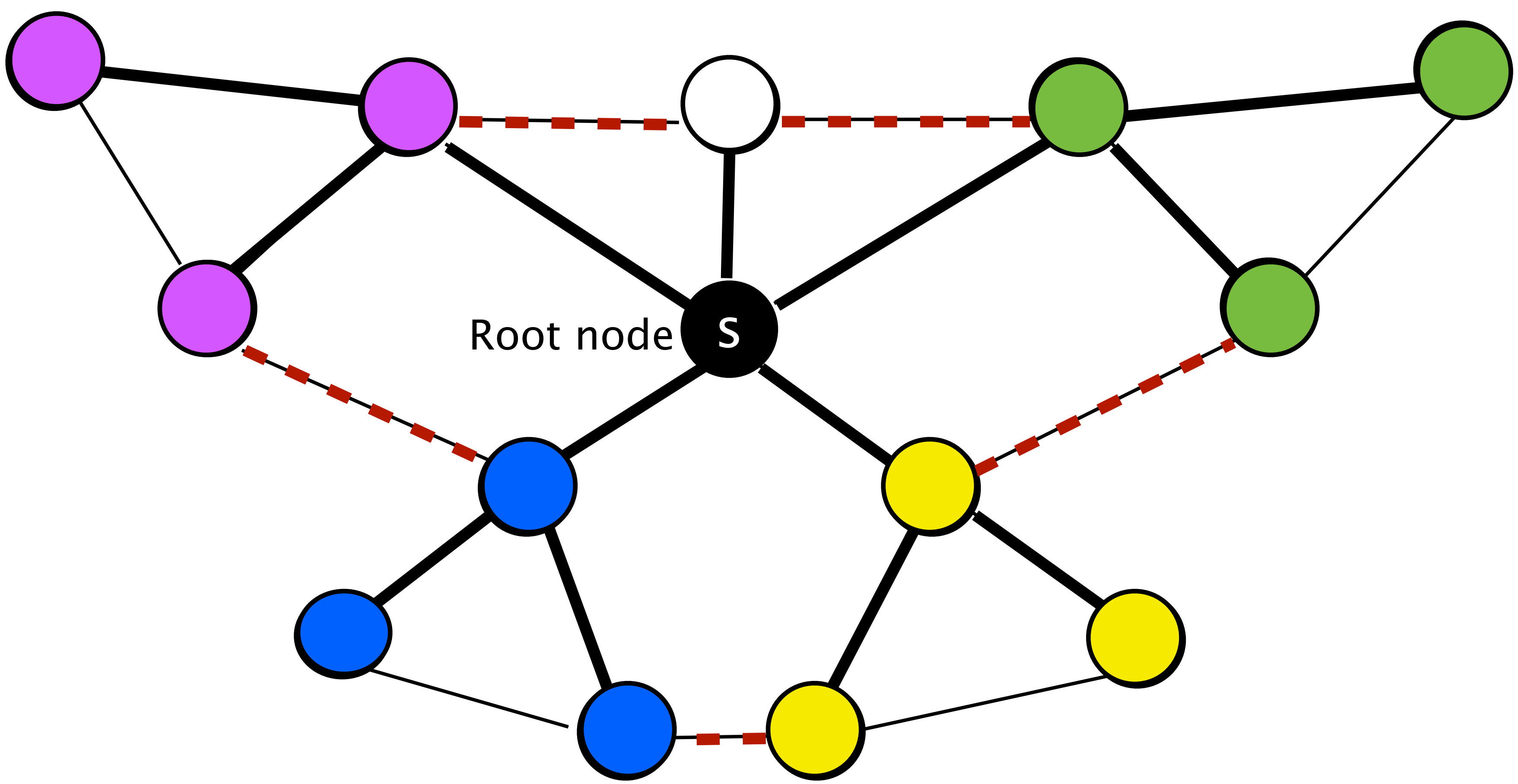
1. SPT edges: links belonging to the SPT
2. Transverses edges: links between branches
3. Internal edges : links between nodes of the same branch



Graph terminology

- The edges of a graph can be partitioned into 3 categories (considering both directions):

1. SPT edges: links belonging to the SPT
2. Transverses edges: links between branches
3. Internal edges : links between nodes of the same branch

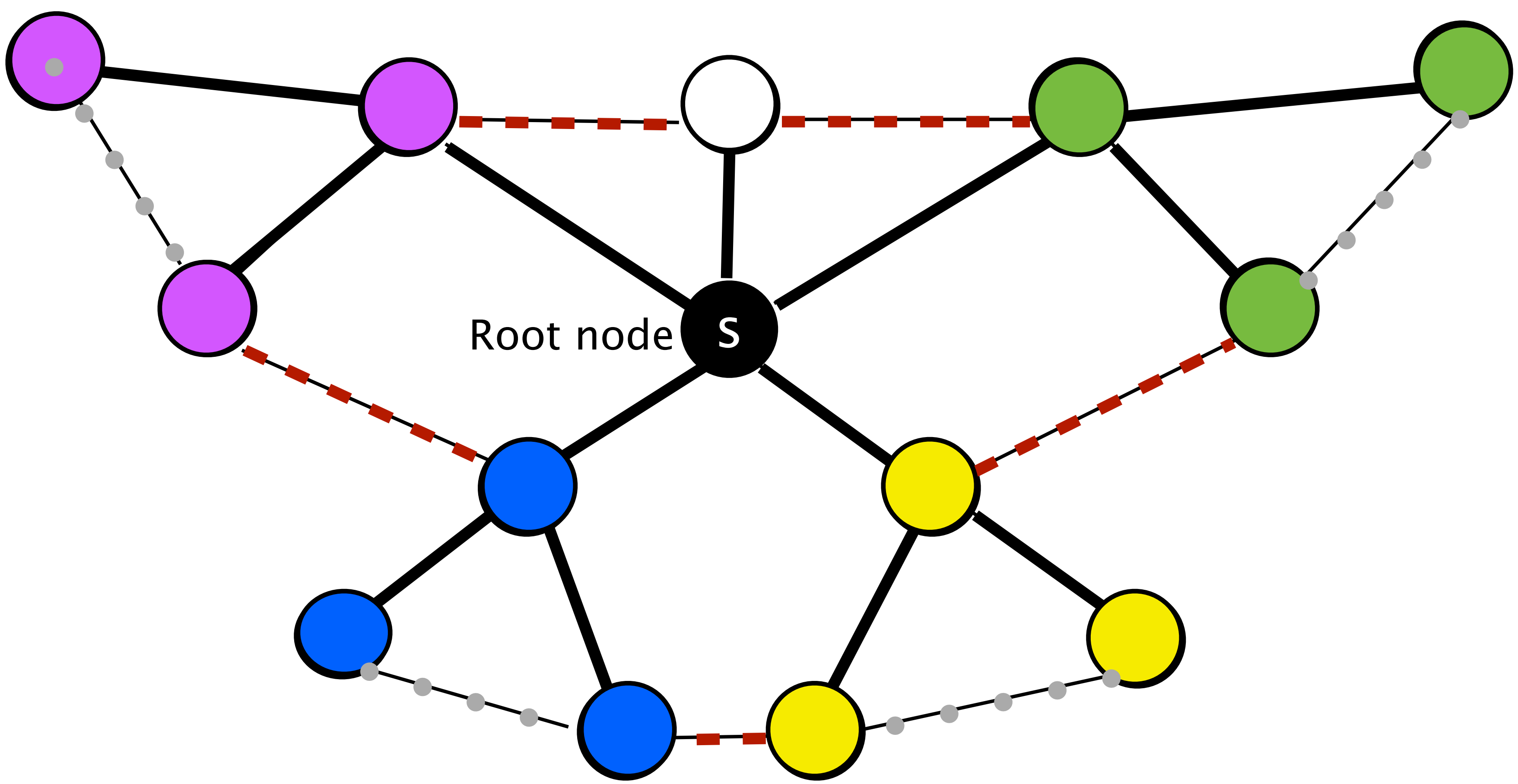


5 branches: 5 subgraphs, each containing a set of primary paths having the same primary next hop

Graph terminology

- The edges of a graph can be partitioned into 3 categories (considering both directions):

1. SPT edges: links belonging to the SPT
2. Transverses edges: links between branches
3. Internal edges : links between nodes of the same branch

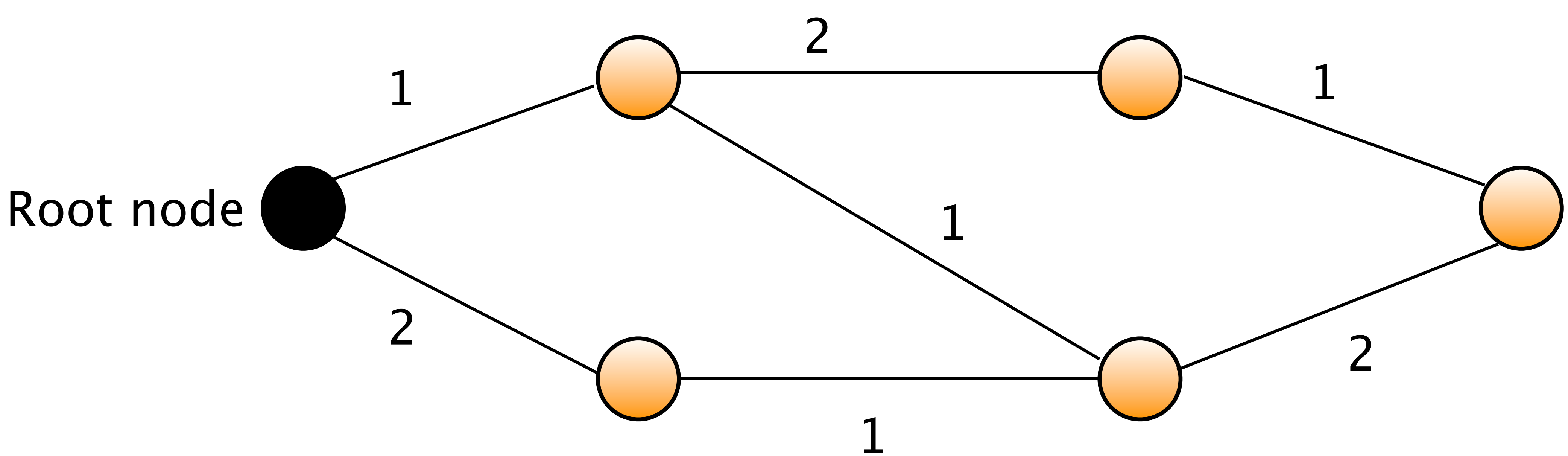


5 branches: 5 subgraphs, each containing a set of primary paths having the same primary next hop

Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

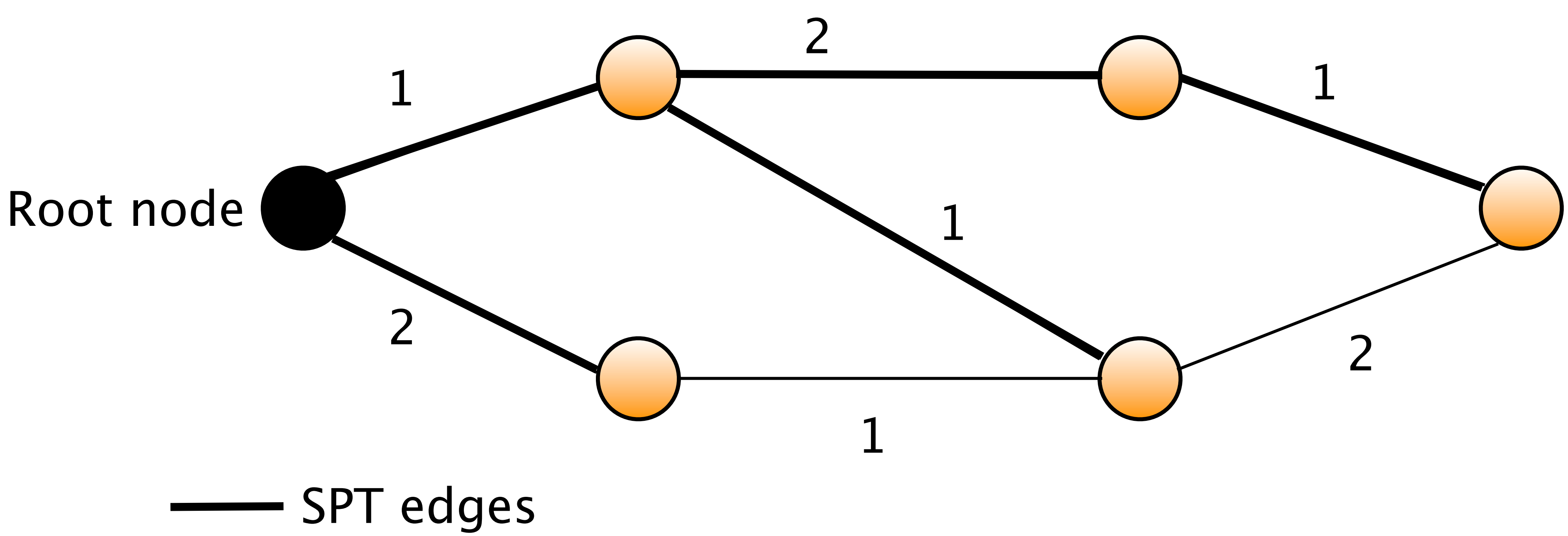
  1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
  2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
  3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

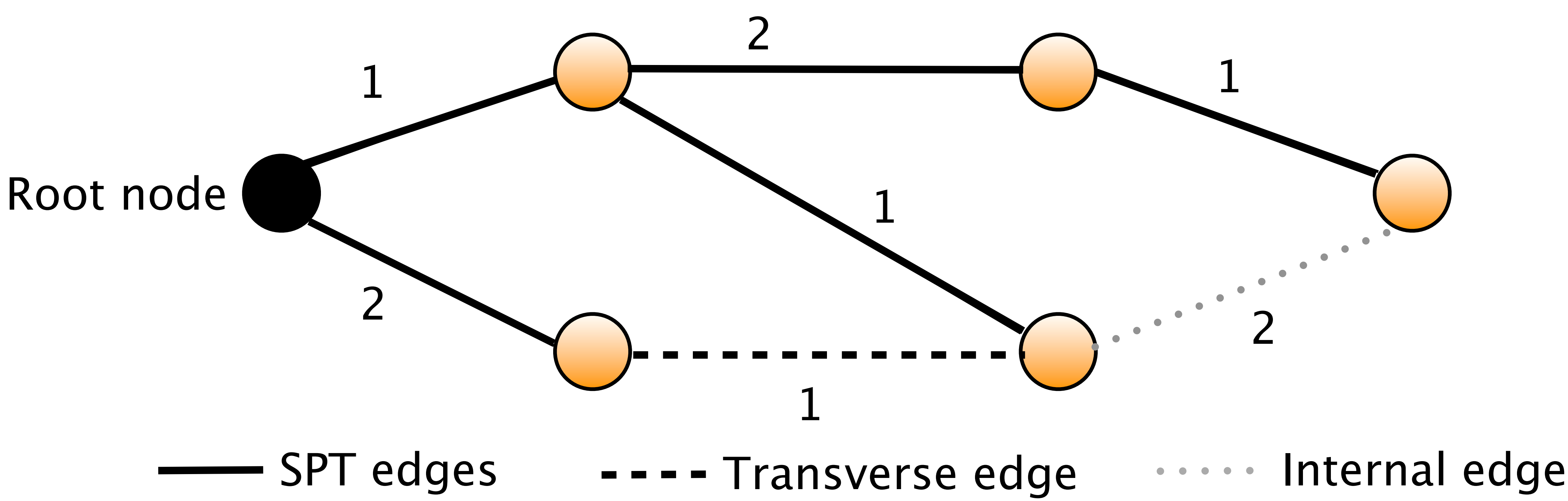
  1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
  2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
  3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

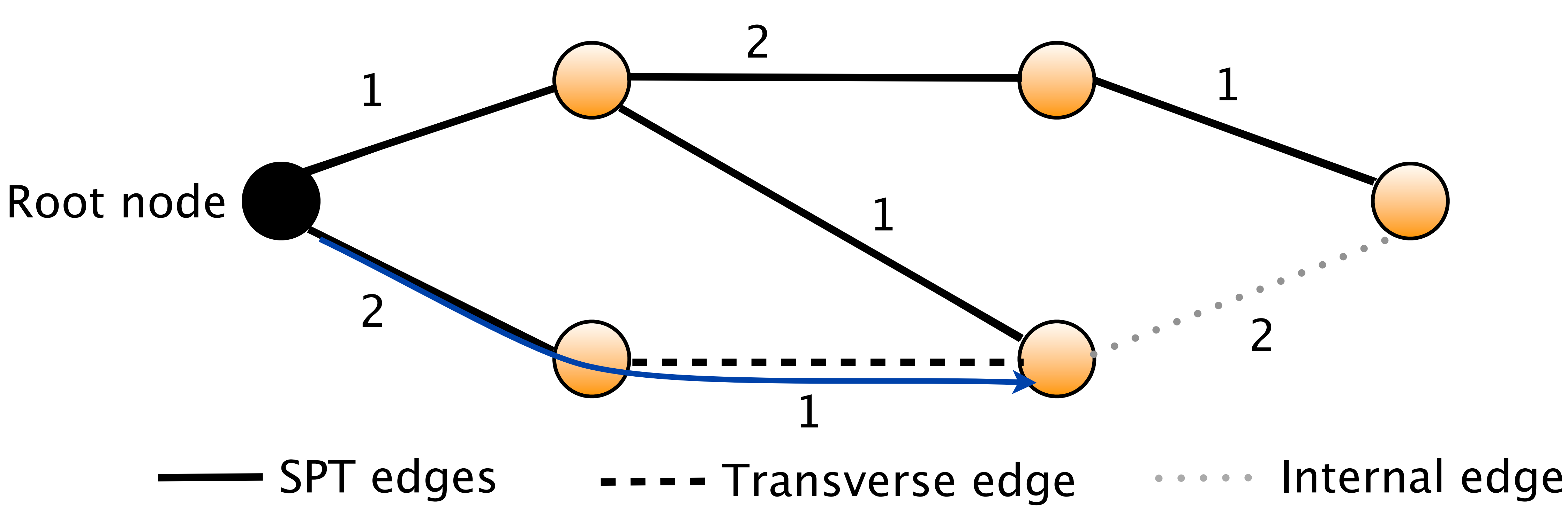
  1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
  2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
  3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

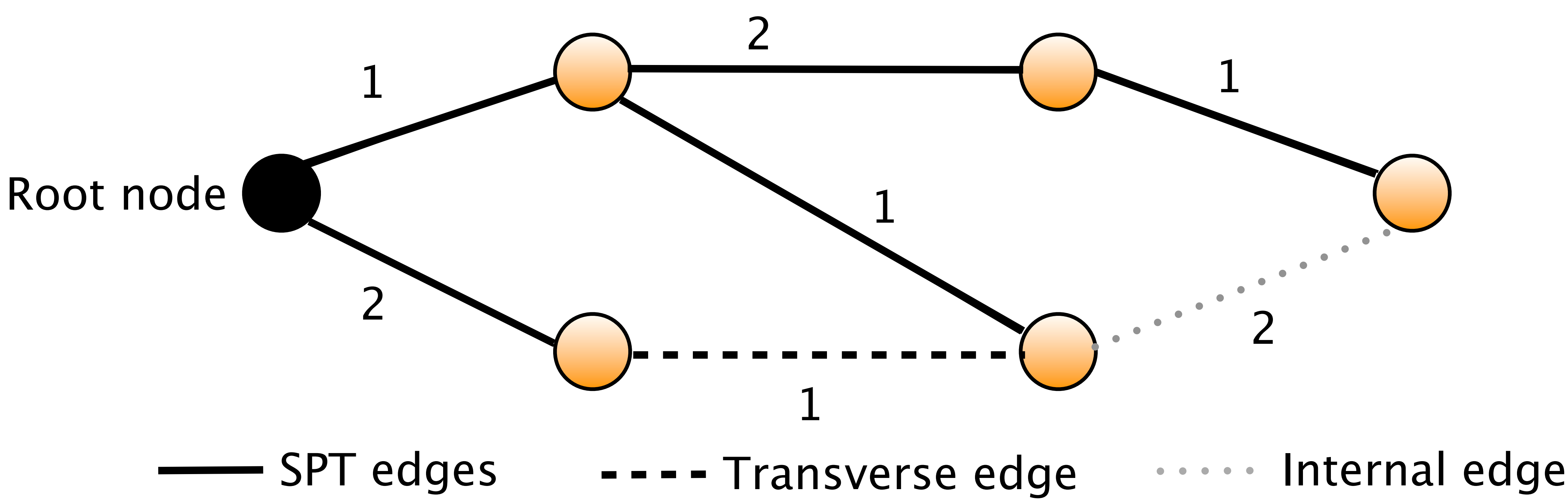
  1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
  2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
  3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

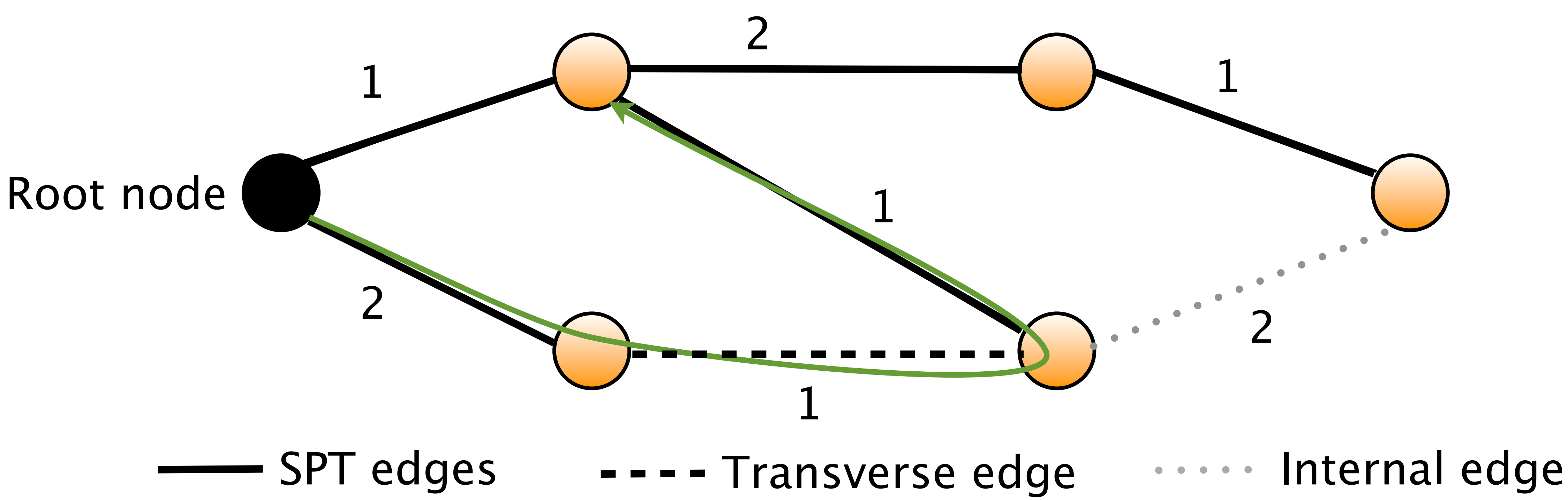
  1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
  2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
  3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

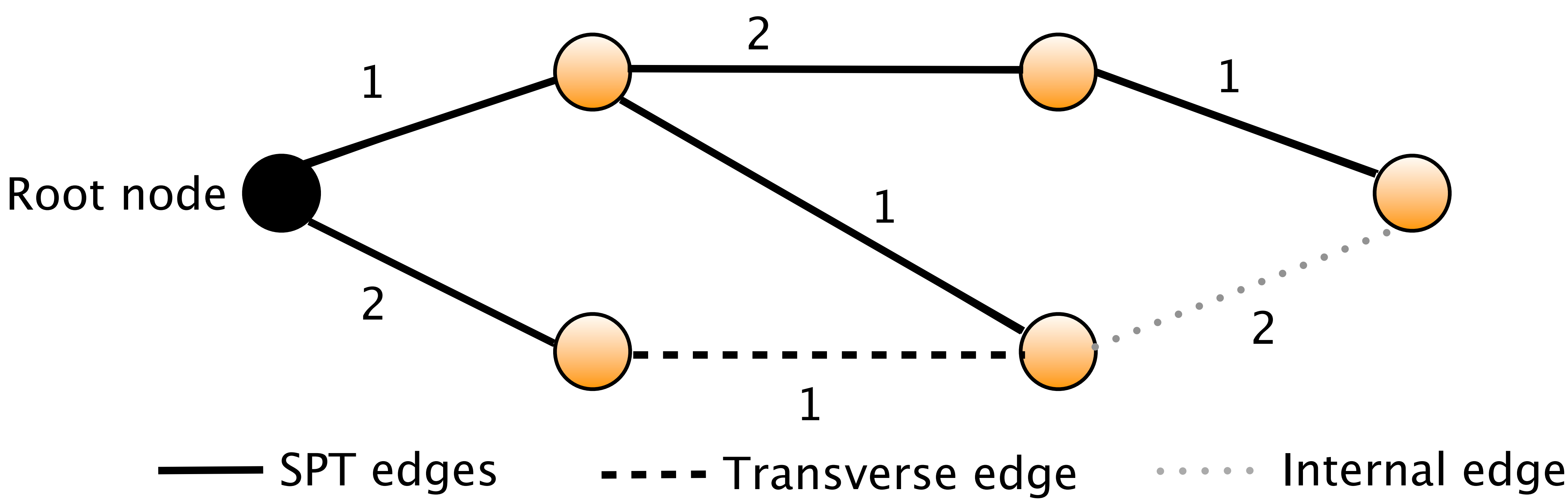
1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

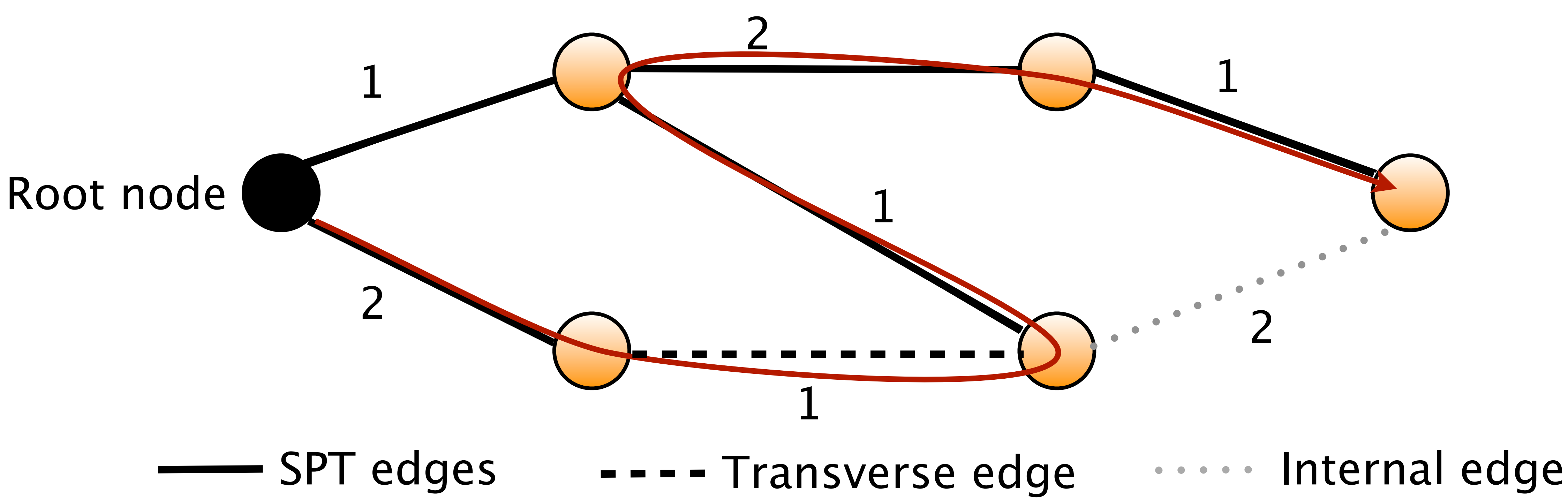
    1.**Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
    2.**Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
    3.**Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Path terminology

- A **1-transverse path** is a path with 1 transverse edge and no internal edges, it can take three forms:

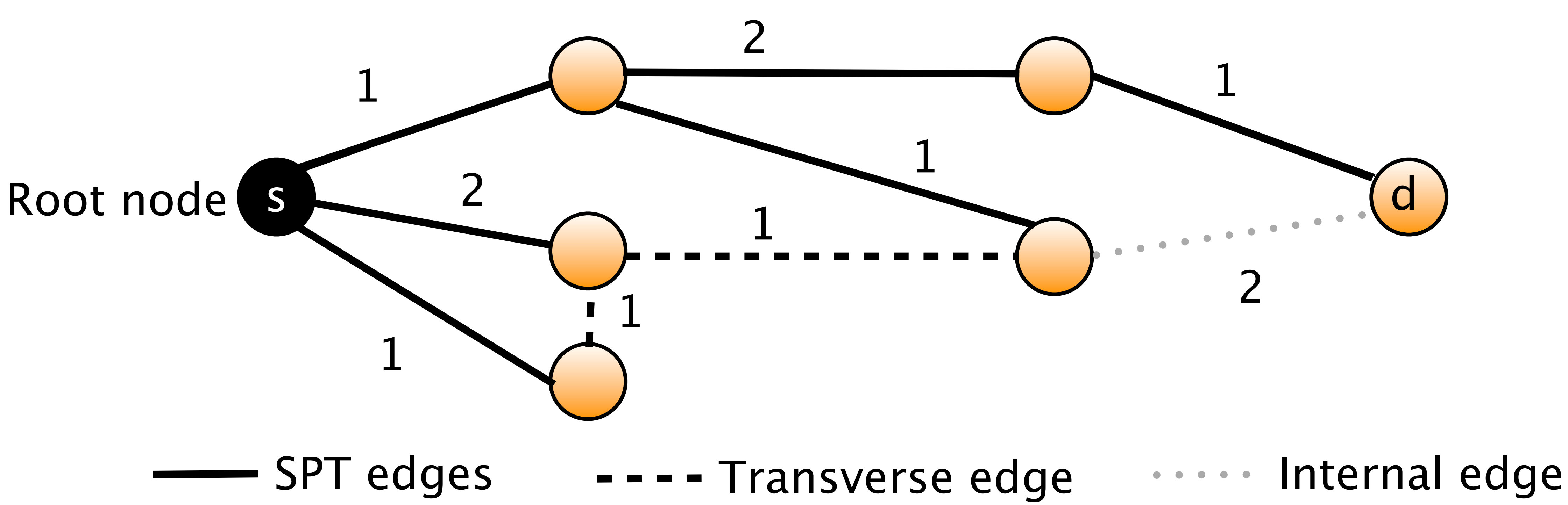
1. **Simple transverse path**: A best path (SPT edges)+ 1 transverse edge
2. **Backward transverse path**: A simple transverse path + n backward SPT edges (n>0)
3. **Forward transverse path**: [A simple transverse path or a backward transverse path] + n forward SPT edges (n>0)



Transverse path properties

*Lemma 1: If there exists an alternate path linking a given pair (s,d), then there exists **a path with only one transverse edge** which is (one of) the shortest alternate path linking s and d*

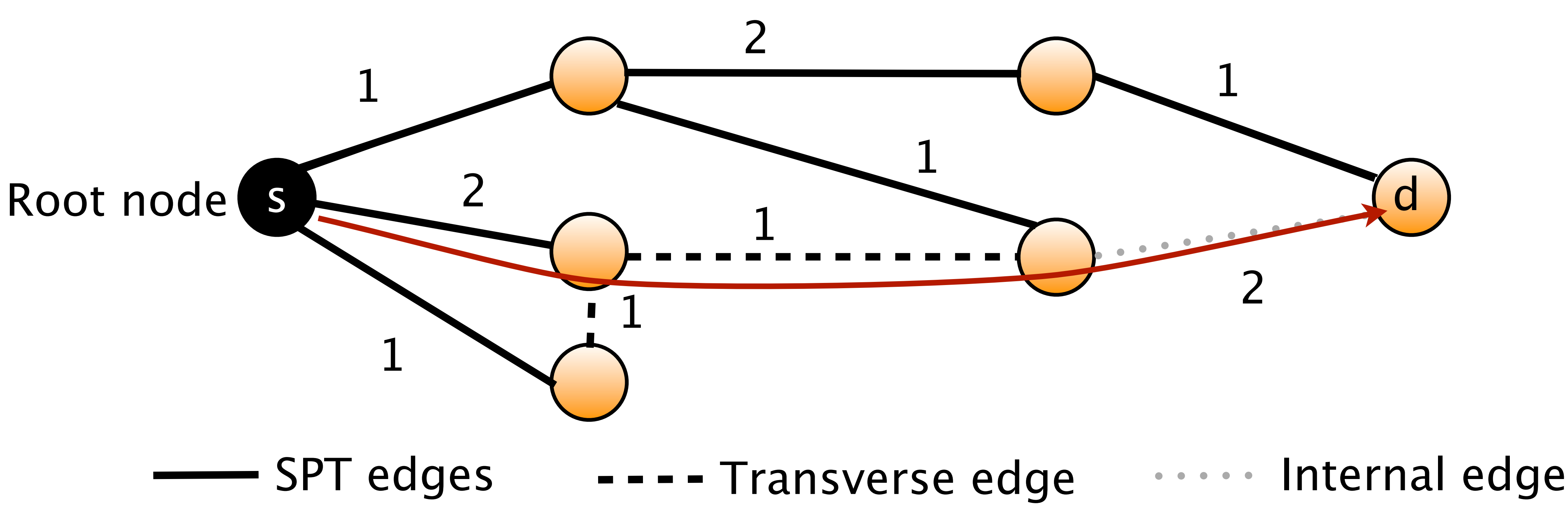
*Lemma 2: If there exists a path with one transverse edge linking a given pair (s,d) and if the existence of edges is symmetric, then there exists **a 1-transverse path** linking s and d*



Transverse path properties

*Lemma 1: If there exists an alternate path linking a given pair (s,d), then there exists **a path with only one transverse edge** which is (one of) the shortest alternate path linking s and d*

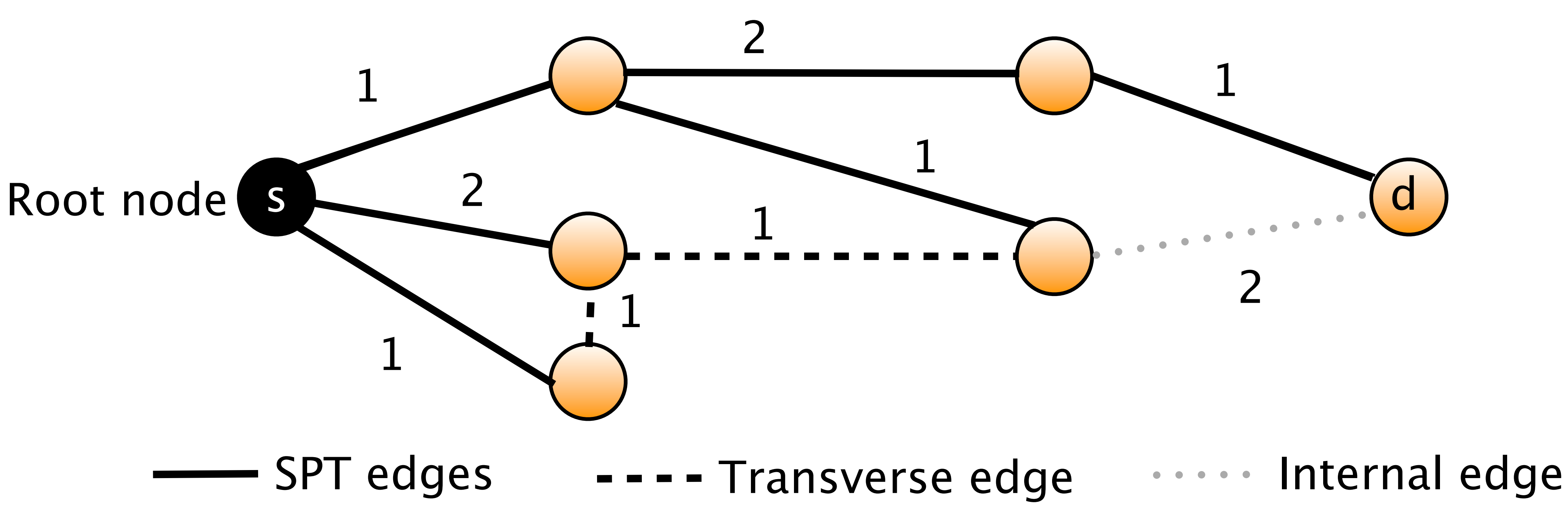
*Lemma 2: If there exists a path with one transverse edge linking a given pair (s,d) and if the existence of edges is symmetric, then there exists **a 1-transverse path** linking s and d*



Transverse path properties

Lemma 1: If there exists an alternate path linking a given pair (s,d), then there exists ***a path with only one transverse edge*** *which is (one of) the shortest alternate path linking s and d*

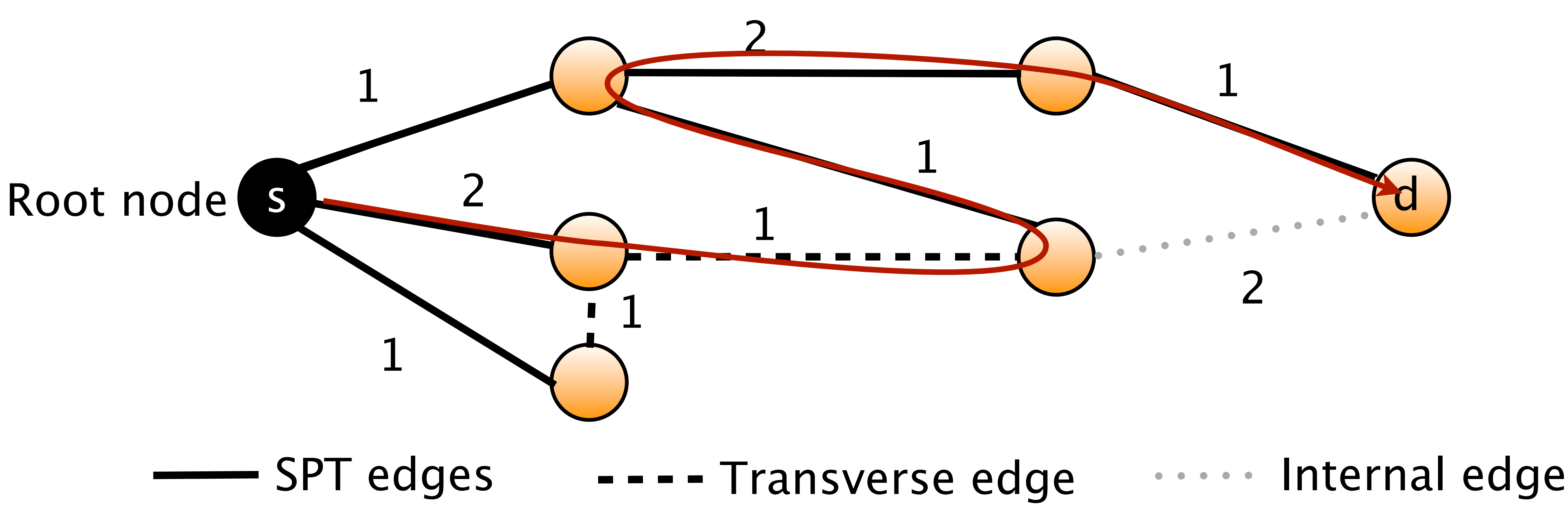
Lemma 2: If there exists a path with one transverse edge linking a given pair (s,d) and if the existence of edges is symmetric, then there exists ***a 1-transverse path*** *linking s and d*



Transverse path properties

Lemma 1: If there exists an alternate path linking a given pair (s,d), then there exists ***a path with only one transverse edge*** *which is (one of) the shortest alternate path linking s and d*

Lemma 2: If there exists a path with one transverse edge linking a given pair (s,d) and if the existence of edges is symmetric, then there exists ***a 1-transverse path*** *linking s and d*



The mDT algorithm

- mDT is a variant of the Dijkstra Transverse (DT) algorithm:
- It computes all 1-transverse paths
  ➠ **at least 2 candidate paths for all pairs (src,dst) if the graph is 2-edge connected**
- It computes all best equal cost paths and all paths with one internal edge (...whereas DT does not)
- Its time complexity, $O(k|E|+|N|log|N|+k|N|)$, is slightly greater than the one of DT, $O(|E|+|N|log|N|+k|N|)$, but lower than the one of kD, $O(k(E+|N|log|N|))$
- By design, mDT computes candidate alternate paths whose costs are close to the best one (➠ loop-free rule)

mDT basics

- <u>Two consecutive phases:</u>

1. A enhanced version of the Dijkstra algorithm to compute:

➡Shortest paths (primary next hops)
➡All alternate equal shortest paths (including paths with n transverse edges, n>0, similarly to ECMP)
➡Best simple transverse paths per neighbor
➡Best path with one internal edge per neighbor
★*Complexity : $O(|N|log|N|+k|E|)$*

2. A backward/forward composition algorithm to compute:

➡Best backward transverse paths per neighbor
➡Best forward transverse paths per neighbor
★*Complexity : $O(k|N|)$*

- mDT computes a matrix containing an upper-bound on the cost for each destination and via each neighbor node

- Each entry of the matrix corresponds to a best transverse path implicitly recorded as a triplet:
cost/neighbor/destination

Evaluation results

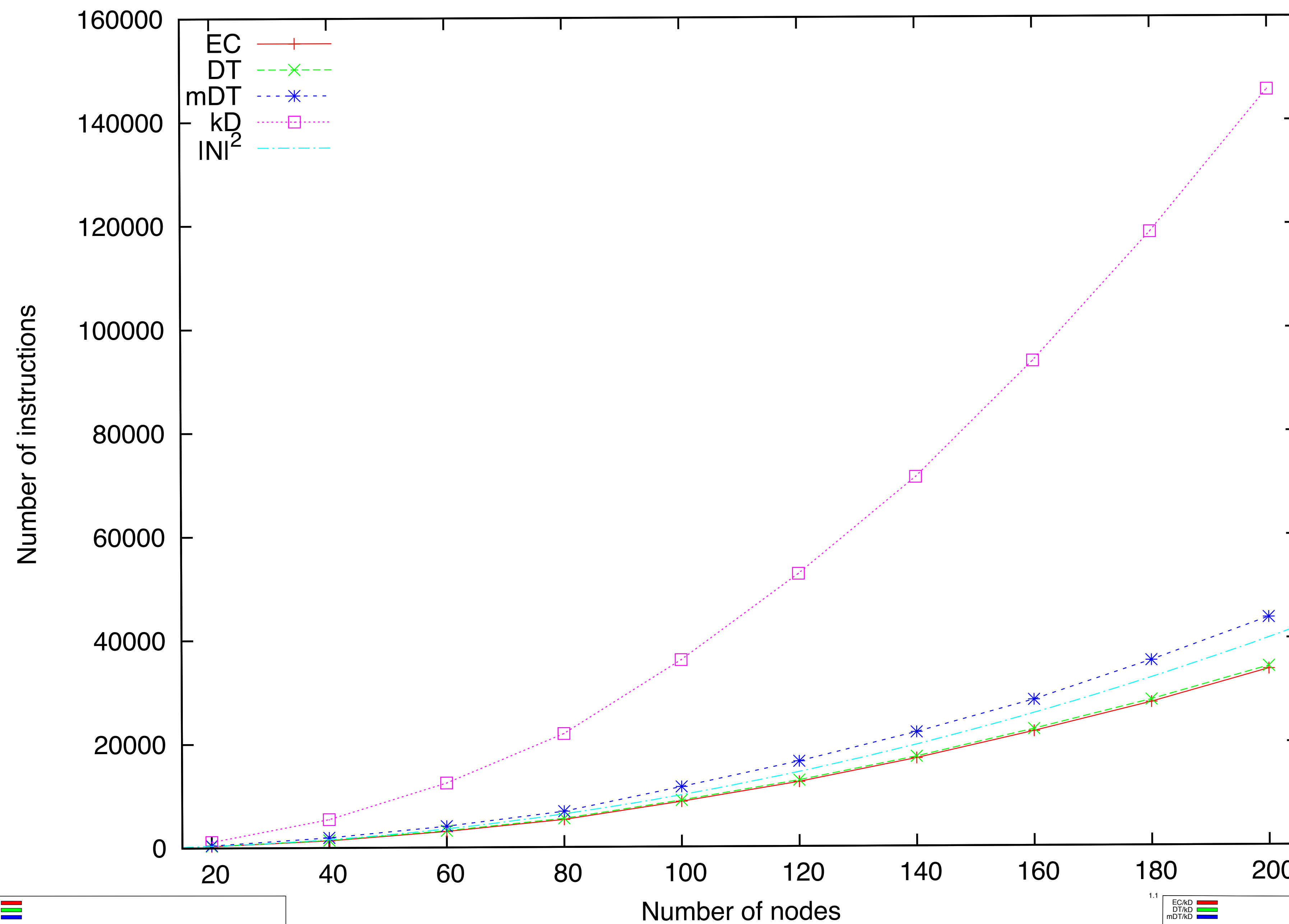
- Analysis setup:

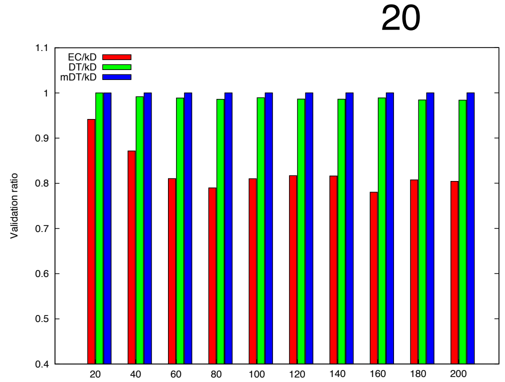
1. NS-2 simulator implementation (available online)
2. Topologies from IGEN generator, Rocketfuel data set and real ones
3. Comparison between ECMP, kD, DT and mDT algorithm
4. Loop-free rule : *the downstream criteria*

- Analysis criteria:

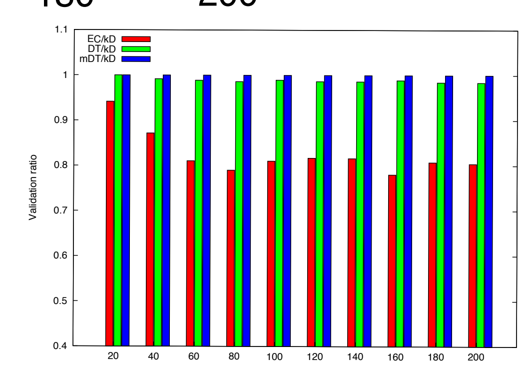
1. Time complexity (instructions needed to manipulate the Priority Queue: extract_min, delete_key, update_key)
2. Number of candidate next hops
3. Number of validated next hops (loop-free)

Time complexity (array list evaluation)

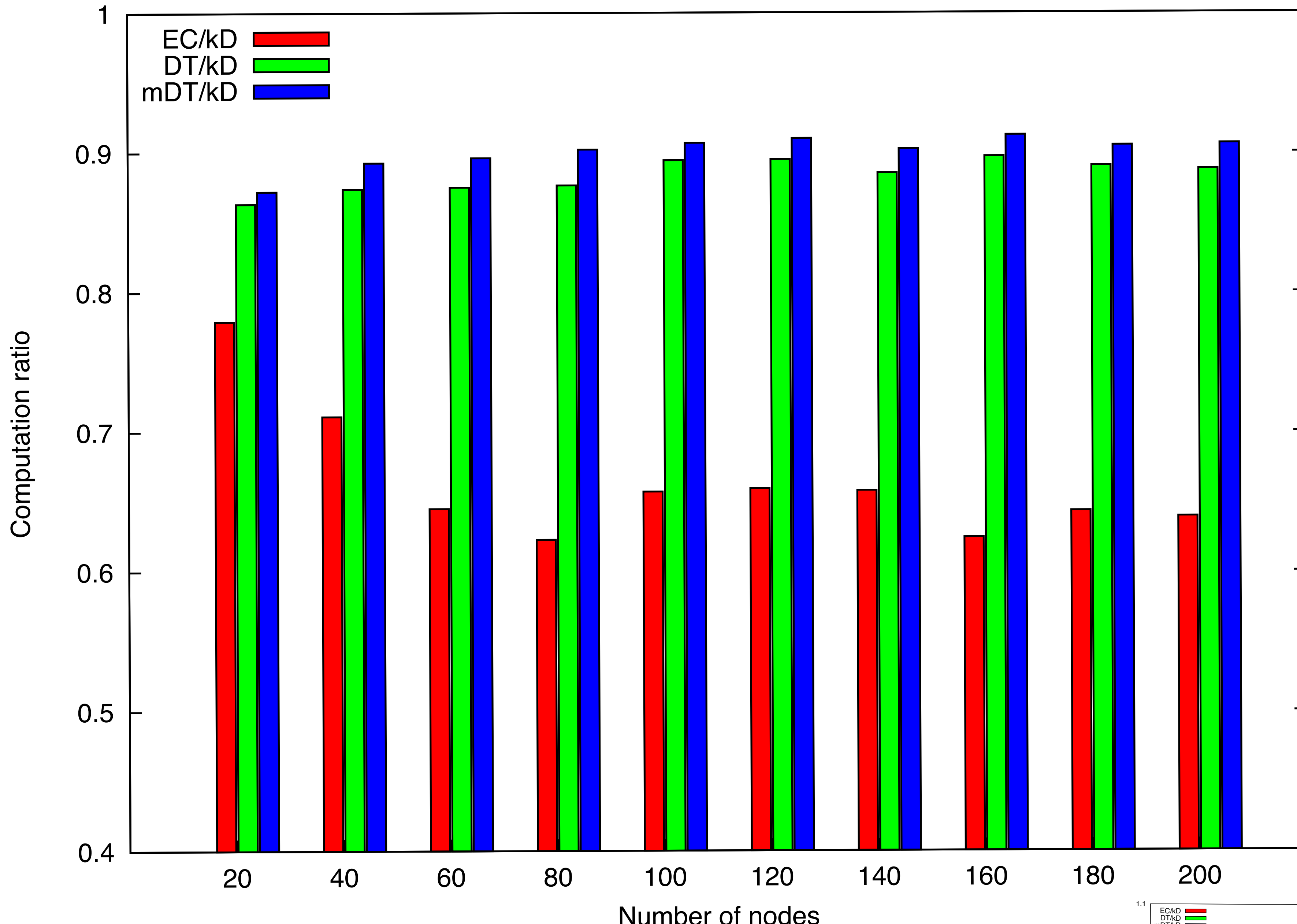




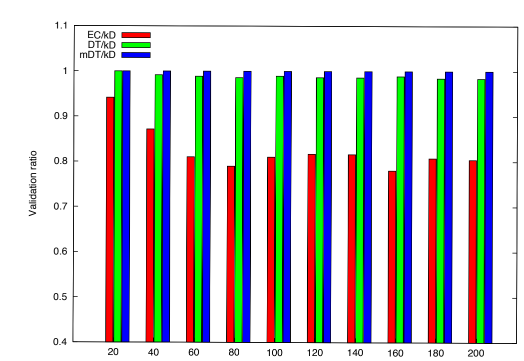
Generated topologies results



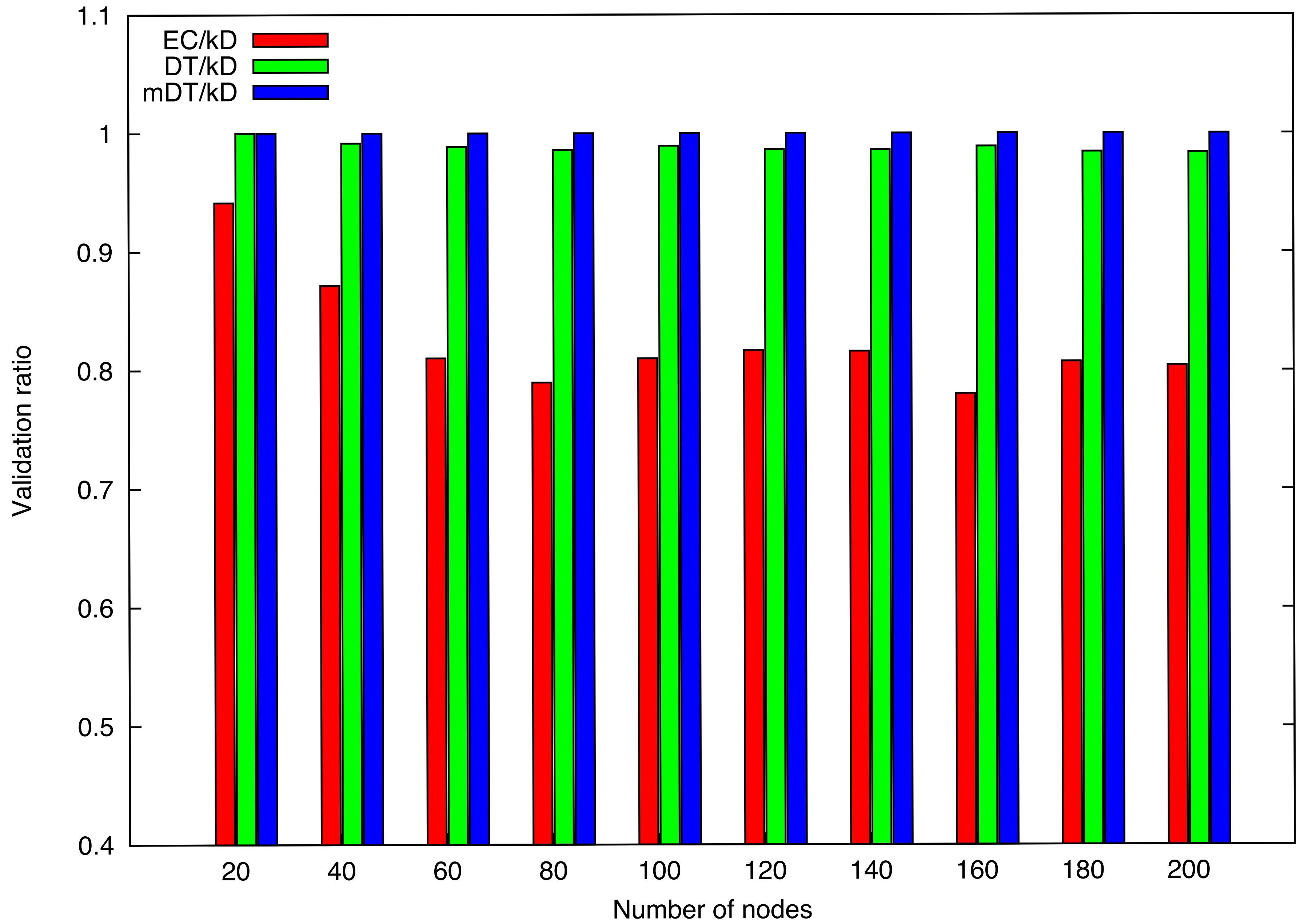
Candidate next hops



Generated topologies results



Loop free next hops



Generated topologies results

Real and inferred topologies

Network name	Size		Candidate next hops				Validated next hops				Number of operations			
			mean	ratio/kD (%)			mean	ratio/kD (%)			mean	ratio/kD (%)		
	$\|N\|$	$\|E\|$	kD	EC	DT	mDT	kD	EC	DT	mDT	kD	EC	DT	mDT
ISP1	25	50	1.46	76	97	97	1.10	97	100	100	489	60	66	75
ISP2	50	200	3.58	43	93	97	1.79	69	89	94	6730	30	32	32.5
ISP3	110	350	2.70	55	89	92	1.45	82	97	99	8079	38	41	43.5
ISP4	210	880	3.73	44	86	88	1.81	72	96	99	41747	27	28	31
Exodus	79	294	3.58	44	88	96	1.73	58	94	99	5569	29	34	37
Ebone	87	322	3.49	46	90	96	1.76	77	93	99	9698	30	33	36
Telstra	104	304	2.30	72	92	95	1.30	90	98	99	6526	54	57	59
Above	141	748	5.29	34	86	97	2.50	58	89	99	40143	18.5	20	23
Tiscali	161	656	3.68	54	91	97	1.97	74	92	97	31044	27	29	32

➡mDT is able to perform the computation of almost the same number of validated (loop-free) next hops than kD but with a lower time complexity

➡The save in term of time complexity is proportional to the existing physical path diversity, e.g, the more the network is connected, the more the overhead in time complexity of kD becomes useless

Real and inferred topologies

Network name	Size		Candidate next hops				Validated next hops				Number of operations			
			mean	ratio/kD (%)			mean	ratio/kD (%)			mean	ratio/kD (%)		
	$\|N\|$	$\|E\|$	kD	EC	DT	mDT	kD	EC	DT	mDT	kD	EC	DT	mDT
ISP1	25	50	1.46	76	97	97	1.10	97	100	100	489	60	66	75
ISP2	50	200	3.58	43	93	97	1.79	69	89	94	6730	30	32	32.5
ISP3	110	350	2.70	55	89	92	1.45	82	97	99	8079	38	41	43.5
ISP4	210	880	3.73	44	86	88	1.81	72	96	99	41747	27	28	31
Exodus	79	294	3.58	44	88	96	1.73	58	94	99	5569	29	34	37
Ebone	87	322	3.49	46	90	96	1.76	77	93	99	9698	30	33	36
Telstra	104	304	2.30	72	92	95	1.30	90	98	99	6526	54	57	59
Above	141	748	5.29	34	86	97	2.50	58	89	99	40143	18.5	20	23
Tiscali	161	656	3.68	54	91	97	1.97	74	92	97	31044	27	29	32

➡mDT is able to perform the computation of almost the same number of validated (loop-free) next hops than kD but with a lower time complexity

➡The save in term of time complexity is proportional to the existing physical path diversity, e.g, the more the network is connected, the more the overhead in time complexity of kD becomes useless

Real and inferred topologies

Network name	Size		Candidate next hops				Validated next hops				Number of operations			
			mean	ratio/kD (%)			mean	ratio/kD (%)			mean	ratio/kD (%)		
	$\|N\|$	$\|E\|$	kD	EC	DT	mDT	kD	EC	DT	mDT	kD	EC	DT	mDT
ISP1	25	50	1.46	76	97	97	1.10	97	100	100	489	60	66	75
ISP2	50	200	3.58	43	93	97	1.79	69	89	94	6730	30	32	32.5
ISP3	110	350	2.70	55	89	92	1.45	82	97	99	8079	38	41	43.5
ISP4	210	880	3.73	44	86	88	1.81	72	96	99	41747	27	28	31
Exodus	79	294	3.58	44	88	96	1.73	58	94	99	5569	29	34	37
Ebone	87	322	3.49	46	90	96	1.76	77	93	99	9698	30	33	36
Telstra	104	304	2.30	72	92	95	1.30	90	98	99	6526	54	57	59
Above	141	748	5.29	34	86	97	2.50	58	89	99	40143	18.5	20	23
Tiscali	161	656	3.68	54	91	97	1.97	74	92	97	31044	27	29	32

➡mDT is able to perform the computation of almost the same number of validated (loop-free) next hops than kD but with a lower time complexity

➡The save in term of time complexity is proportional to the existing physical path diversity, e.g, the more the network is connected, the more the overhead in time complexity of kD becomes useless

Real and inferred topologies

Network name	Size		Candidate next hops				Validated next hops				Number of operations			
			mean	ratio/kD (%)			mean	ratio/kD (%)			mean	ratio/kD (%)		
	$\|N\|$	$\|E\|$	kD	EC	DT	mDT	kD	EC	DT	mDT	kD	EC	DT	mDT
ISP1	25	50	1.46	76	97	97	1.10	97	100	100	489	60	66	75
ISP2	50	200	3.58	43	93	97	1.79	69	89	94	6730	30	32	32.5
ISP3	110	350	2.70	55	89	92	1.45	82	97	99	8079	38	41	43.5
ISP4	210	880	3.73	44	86	88	1.81	72	96	99	41747	27	28	31
Exodus	79	294	3.58	44	88	96	1.73	58	94	99	5569	29	34	37
Ebone	87	322	3.49	46	90	96	1.76	77	93	99	9698	30	33	36
Telstra	104	304	2.30	72	92	95	1.30	90	98	99	6526	54	57	59
Above	141	748	5.29	34	86	97	2.50	58	89	99	40143	18.5	20	23
Tiscali	161	656	3.68	54	91	97	1.97	74	92	97	31044	27	29	32

➡mDT is able to perform the computation of almost the same number of validated (loop-free) next hops than kD but with a lower time complexity

➡The save in term of time complexity is proportional to the existing physical path diversity, e.g, the more the network is connected, the more the overhead in time complexity of kD becomes useless

Conclusion

- The «DT suite» allows to limit the time complexity of the path computation phase with minimal guarantees:

1. *Graph property:* At least one candidate alternate next hop is computed between a pair (src,dst) if the primary link is not a bridge link
2. *Evaluation results:* mDT performs path diversity results similar to kD (or with the Topkis's algorithm) but with a lower time complexity

➡ mDT can be incrementally deployed to determine a set of paths suitable for hop by hop multi-path forwarding or for local fast reroute schemes

Thank you

Current work

- Develop improved versions of DT to take into account internal edges in paths computation ➠ real alternate best cost calculation
- Thanks to the property of paths containing at most one transverse edge, we are able to:
  - compute the two shortest paths with a distinct first hop
  - compute two valid paths using a given loop-free criteria (DC, LFA)
- To the best of our knowledge, these algorithms are the lowest time complexity procedures existing to compute two (valid) first hop disjoint paths