Communications Network Design

lecture 06

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

Discipline of Applied Mathematics
School of Mathematical Sciences
University of Adelaide

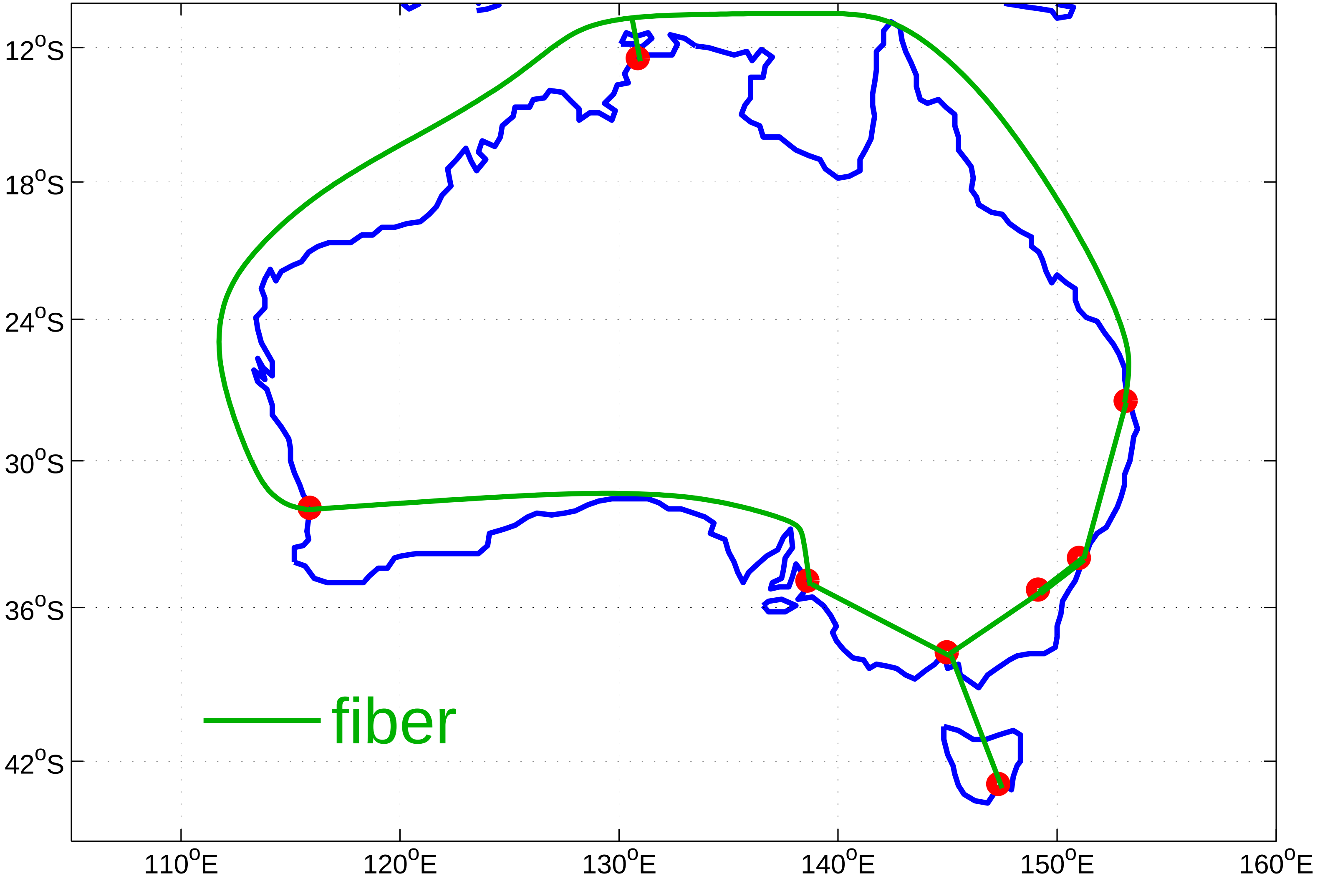
March 23, 2009

Routing

A common approach to routing uses shortest-paths. The canonical algorithm for solving shortest-path routing is Dijkstra's.

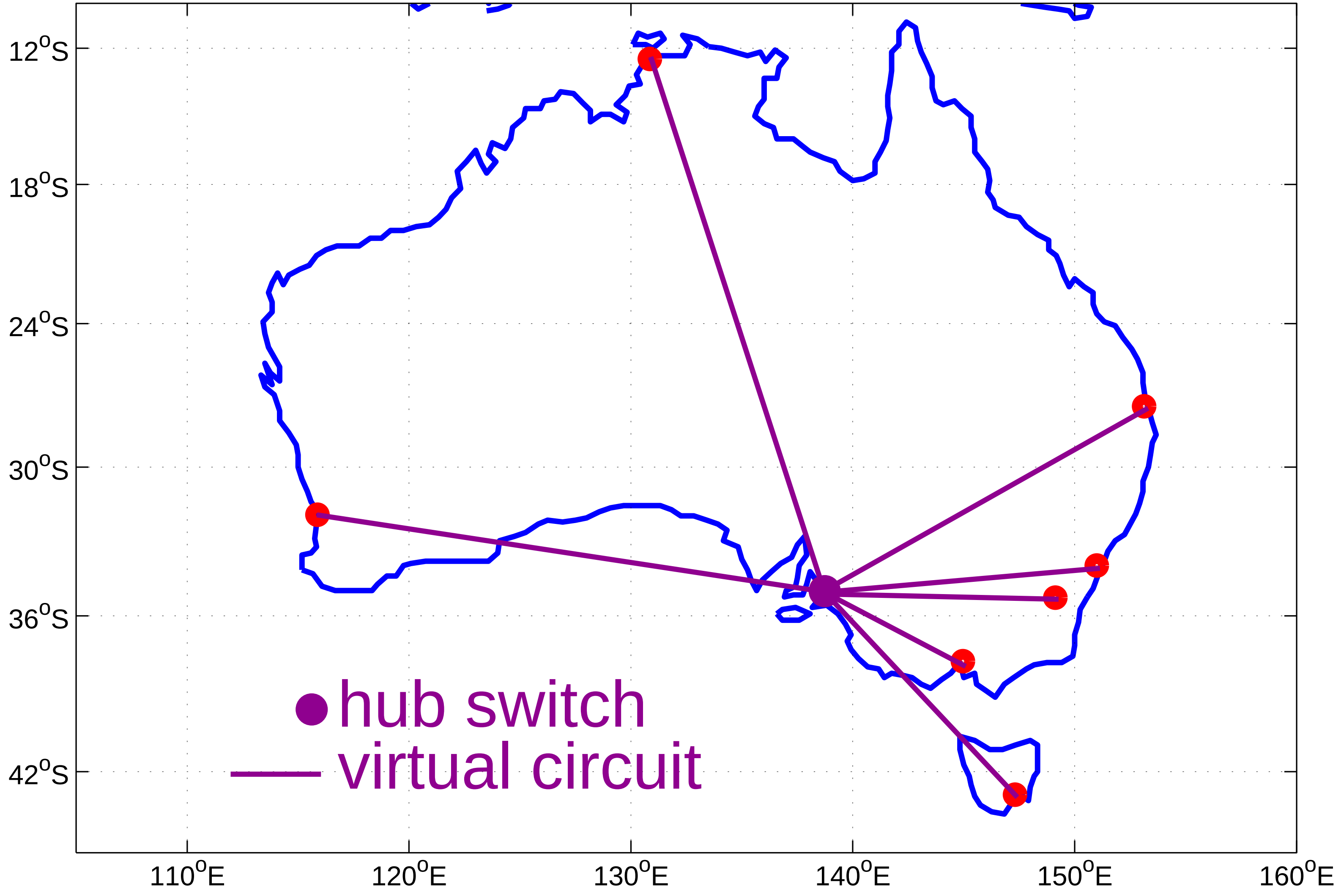
Logical vs Physical Network

Possible physical topology (layer 1)



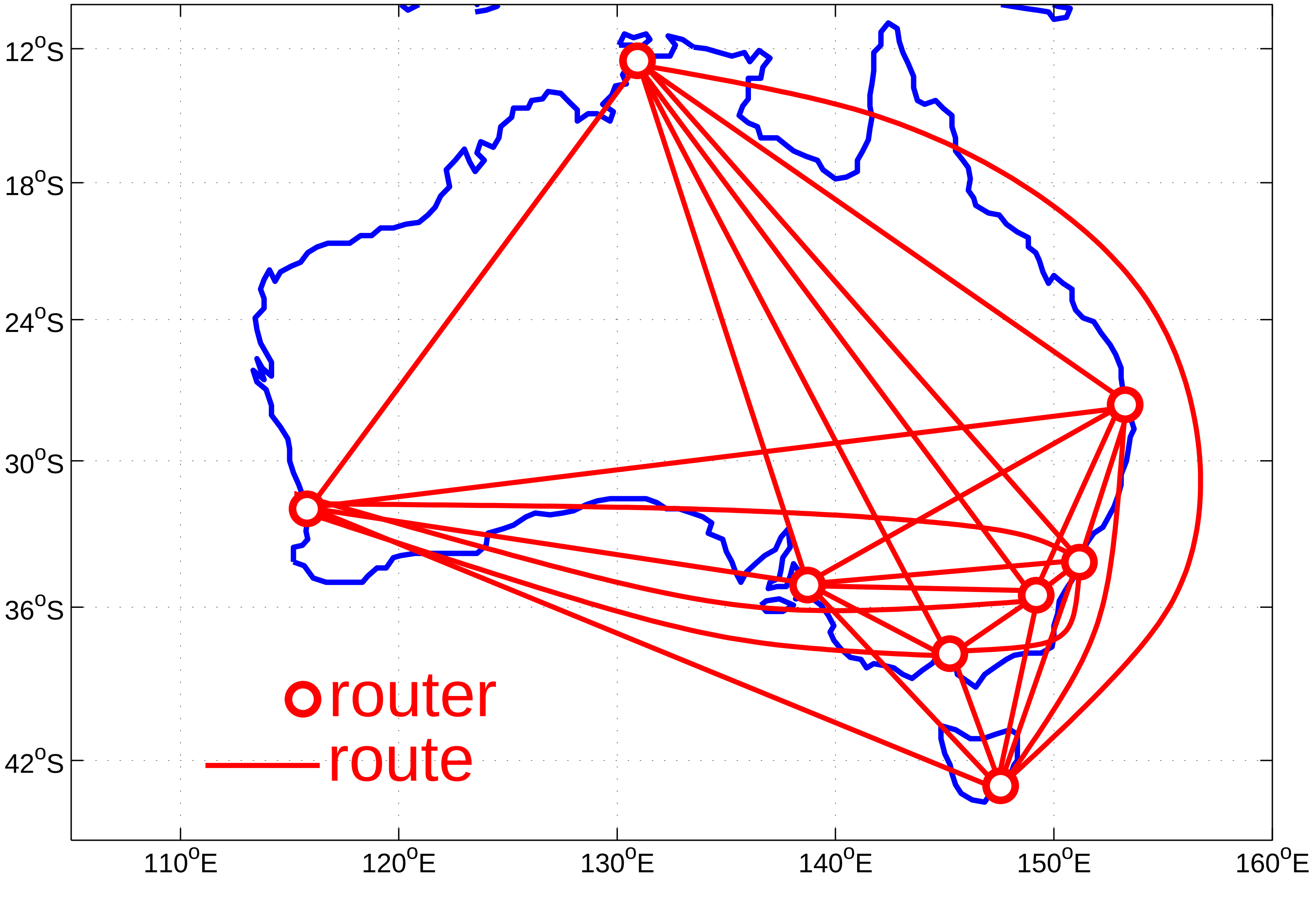
Logical vs Physical Network

Possible logical network topology (layer 2)



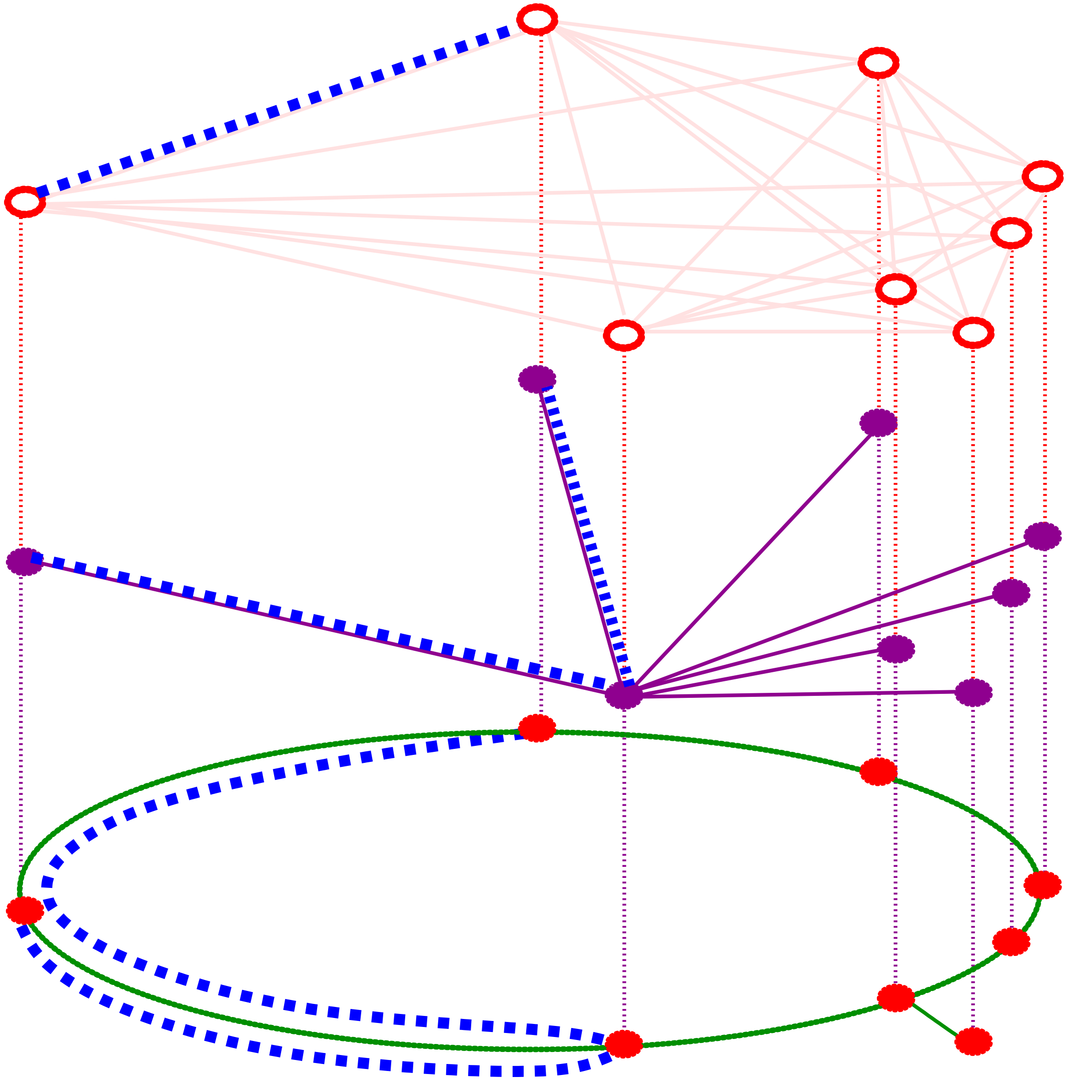
Logical vs Physical Network

Possible logical network topology (layer 3)



Mapping the logical to the physical

Network maps (at one layer) can be quite misleading



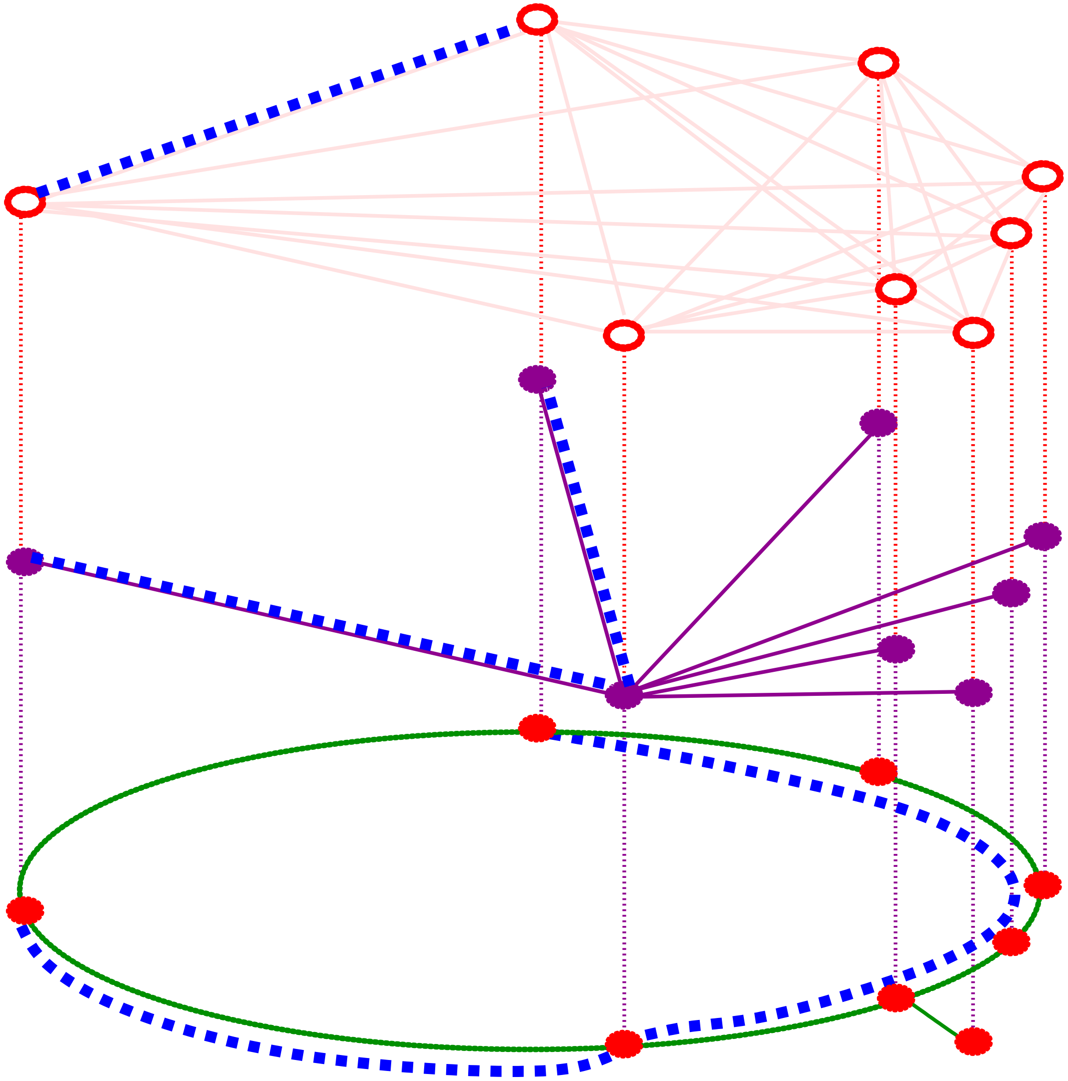
Network
layer 3
routers
packets

Link
layer 2
virtual ciruits
switches

Physical
layer 1
optical fiber
WDM cross connects

Mapping the logical to the physical

Network maps (at one layer) can be quite misleading



Network
layer 3
routers
packets

Link
layer 2
virtual ciruits
switches

Physical
layer 1
optical fiber
WDM cross connects

Circuit switching won't go away

Even for purist IP net-heads

- often circuit switching in lower layers
- G-MPLS – lambda-switching
    - WDM allows multiple wavelengths of light to share a single fiber
    - optical cross-connects switch the light
        - no electronics involved
        - purely optical
    - protocols to set up and tear down optical circuits
- packet forwarding on top of these circuits

Routing

We need a method to map packet routes to links

- called a routing protocol
- several types exist
- we consider (today)
  - link state
  - shortest path
  - IGP (Interior Gateway Protocol)

  routing protocols

Notation and Assumptions

The underlying structure is a graph, with

- a set of nodes $N = \{1, 2, \ldots n\}$ (also called vertex)

$$|N| = n$$

  - a node could be a router, an AS, a PoP, ...
- a set of links $E \subseteq N \times N$ (also called edges)

$$E \subset \{(i, j) : i, j \in N, i \neq j\}$$
$$|E| \leq n(n-1)/2$$

  - a link could be a physical link, logical circuit, ...
- assume the links are undirected, so

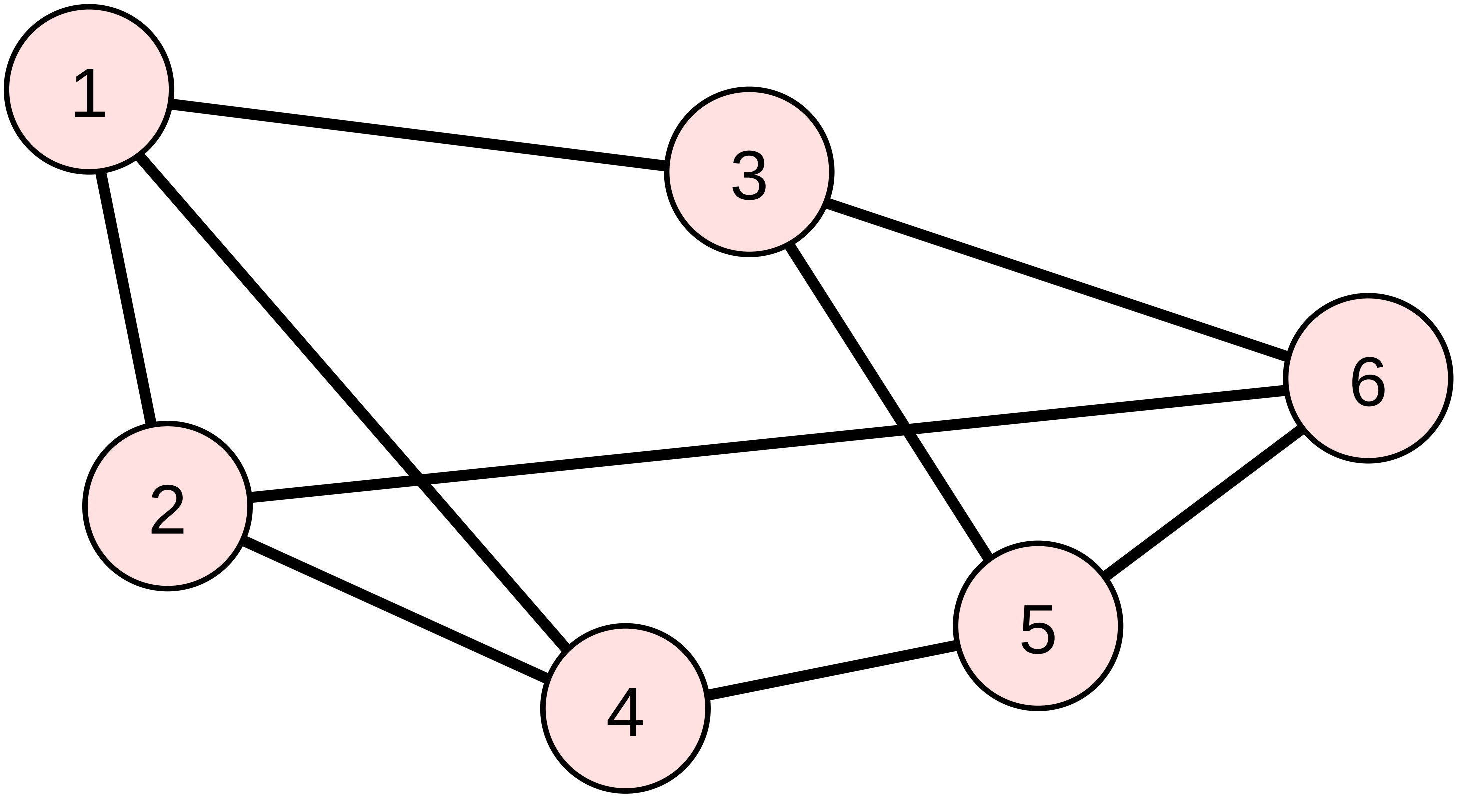
$$(i, j) = (j, i)$$

  - this just makes descriptions easier
  - easily generalized to directed graphs
- The network is defined by the graph, $G(N, E)$

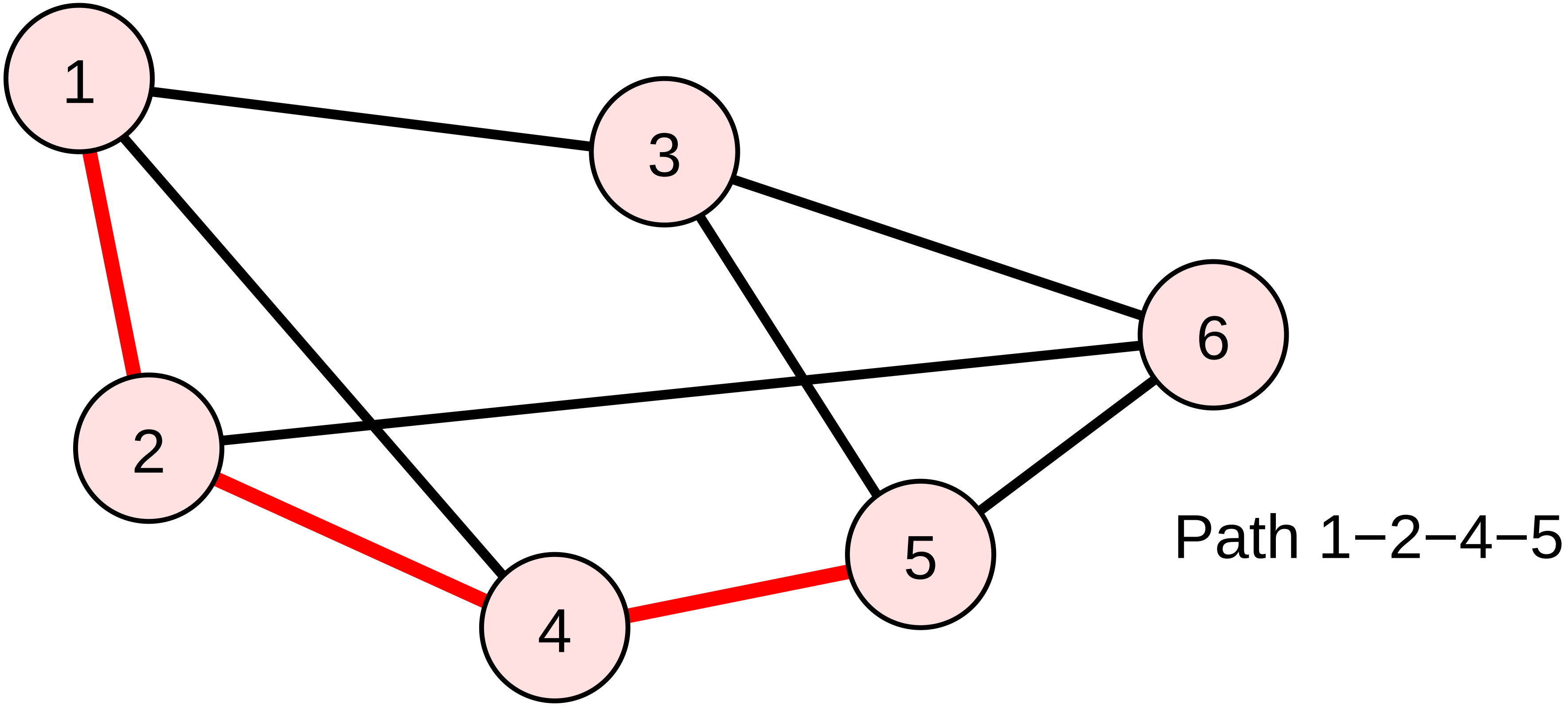
Notation and Assumptions

- Origin-Destination (O-D) pair $(p,q) \in N \times N$
- Let K be the set of all O-D pairs
$$K = \{[p,q] : p,q \in N\}.$$
- Offered traffic between O-D pair (p,q) is t_{pq}
- The set of **paths** in $G(N,E)$ joining an O-D pair (p,q) is denoted P_{pq}.
  - paths are assumed to be a-cyclic
  - e.g. no node is visited twice
  - e.g. loop free
- The set of all paths in $G(N,E)$ is denoted P.
$$P = \cup_{[p,q]\in K} P_{pq}$$

Network Paths

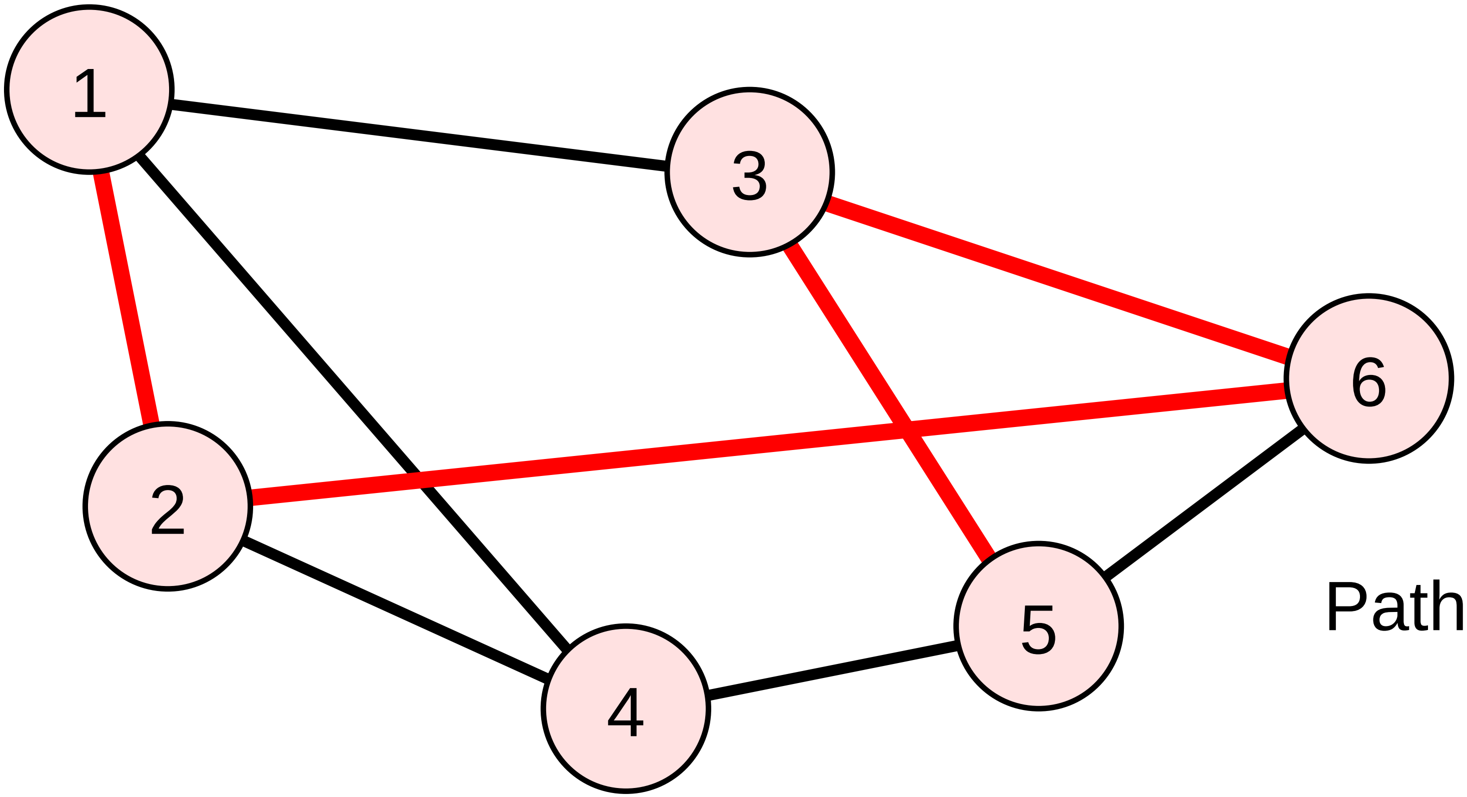


Network Paths



Paths P_{15}: 1-2-4-5

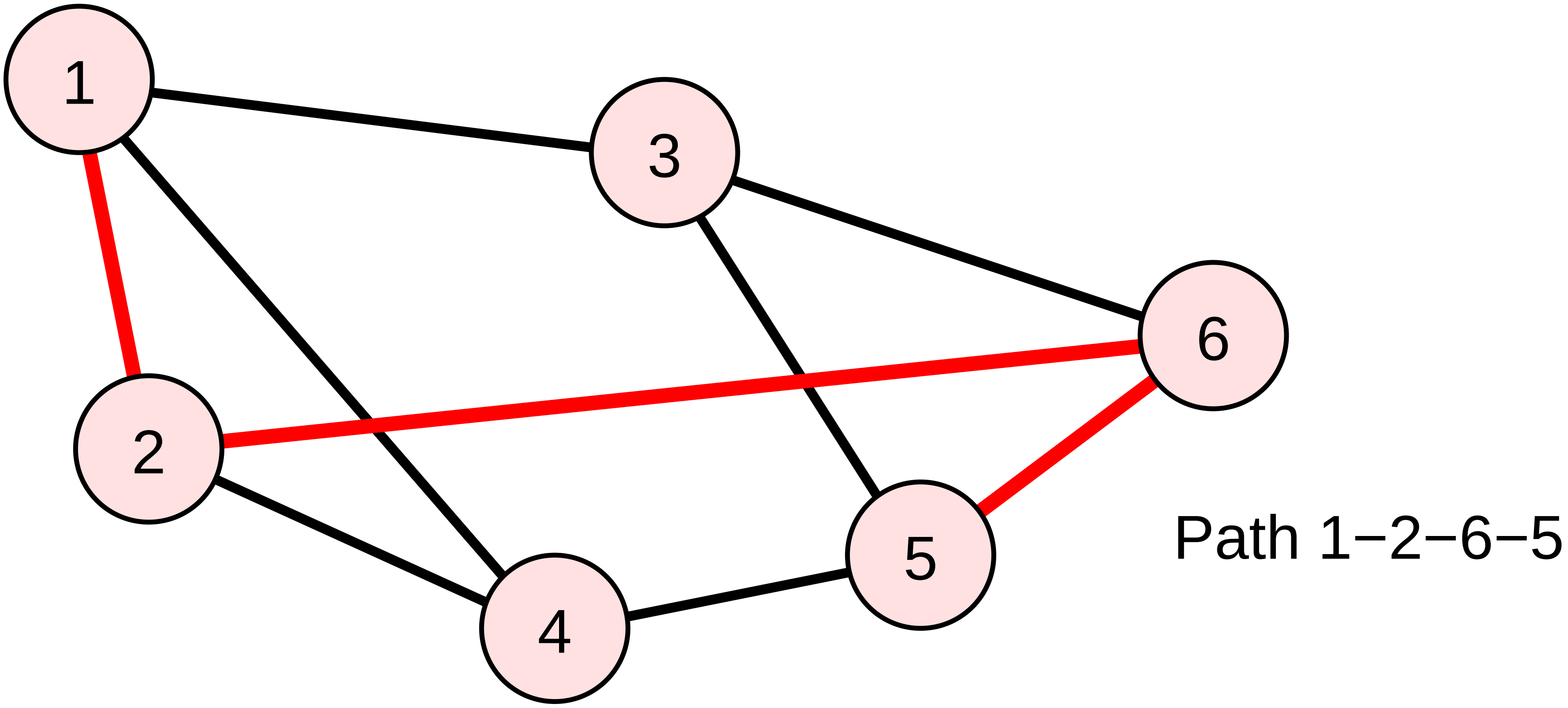
Network Paths



Path 1−2−6−3−5

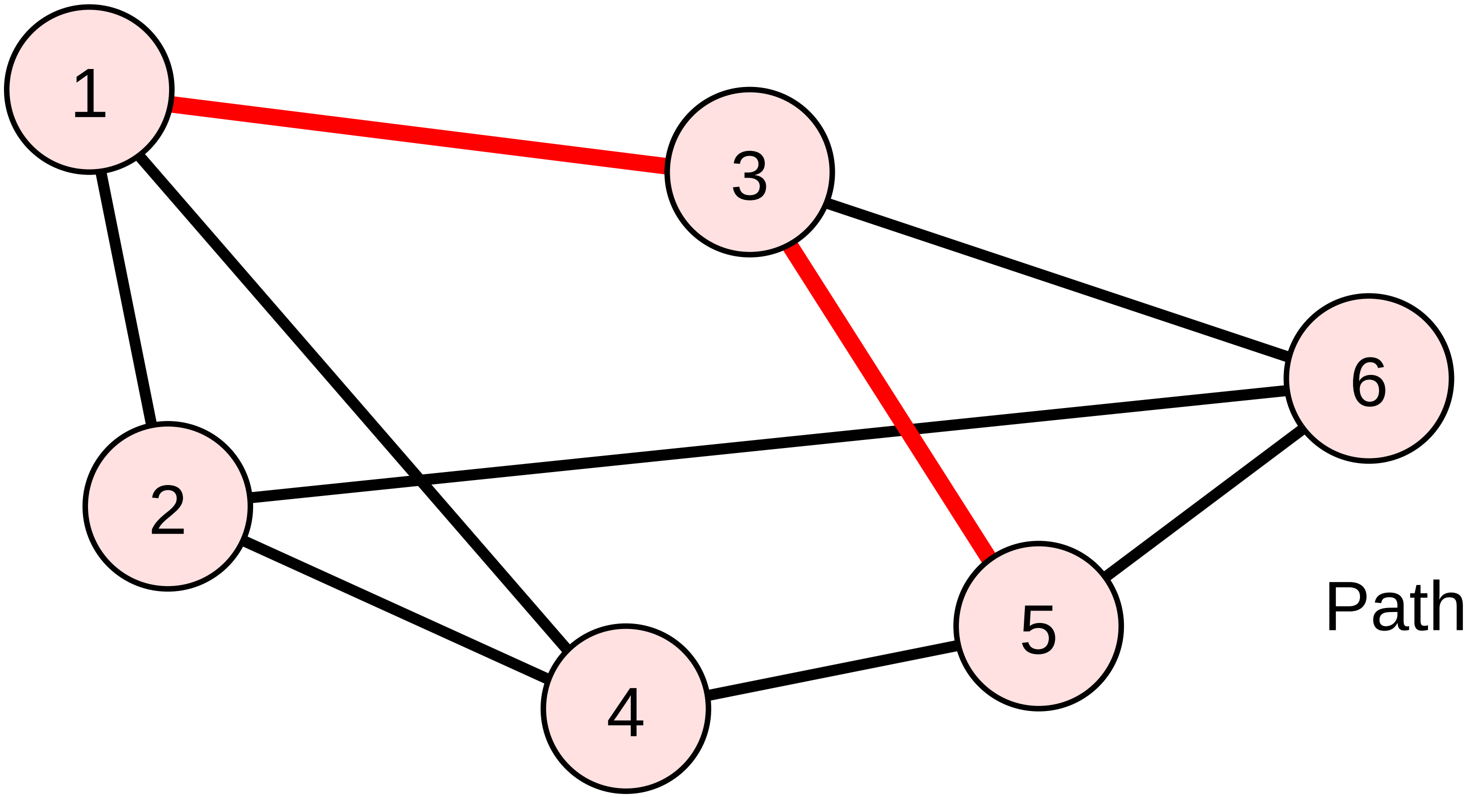
Paths P_{15}: 1-2-4-5, 1-2-6-3-5

Network Paths



Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5

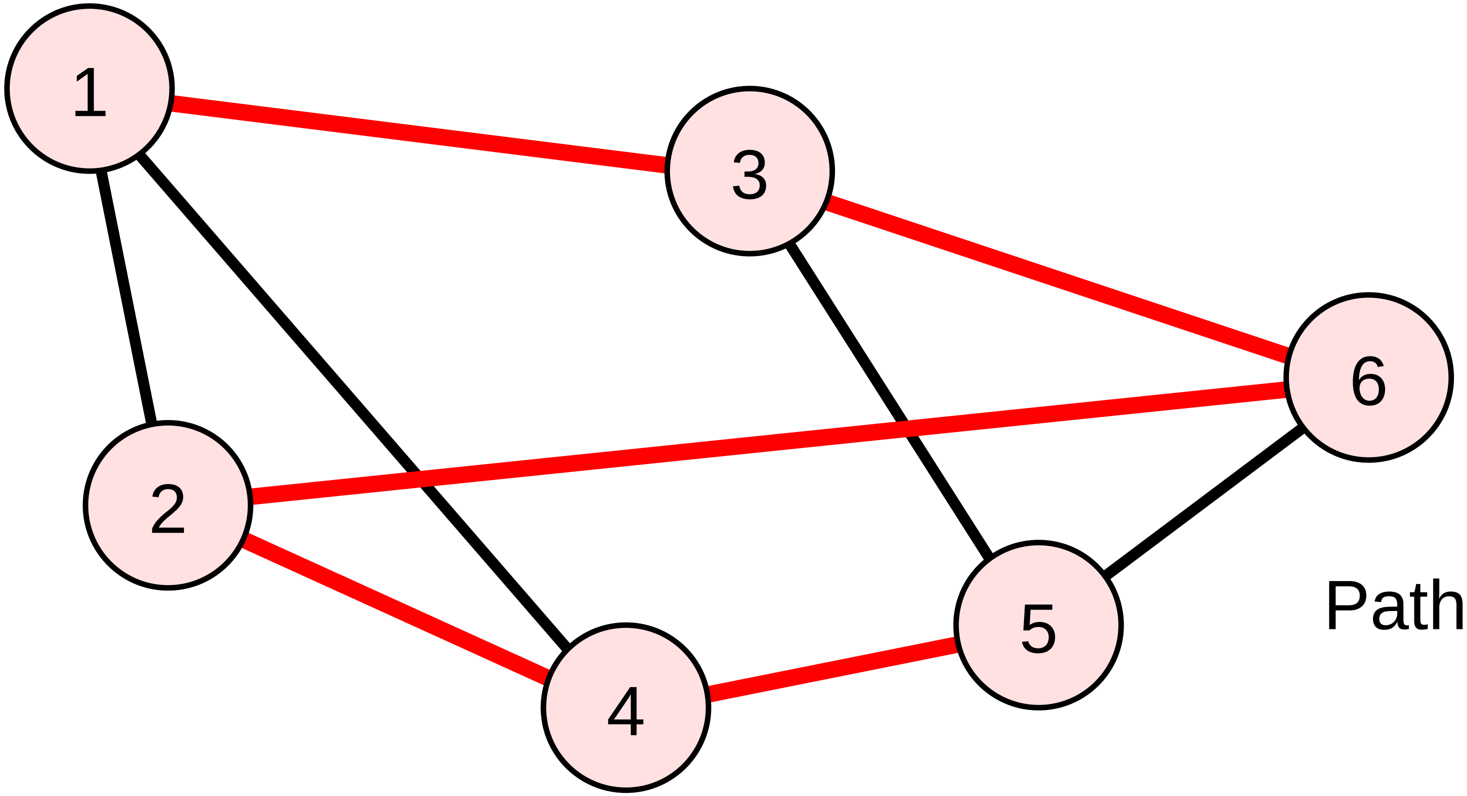
Network Paths



Path 1–3–5

Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5

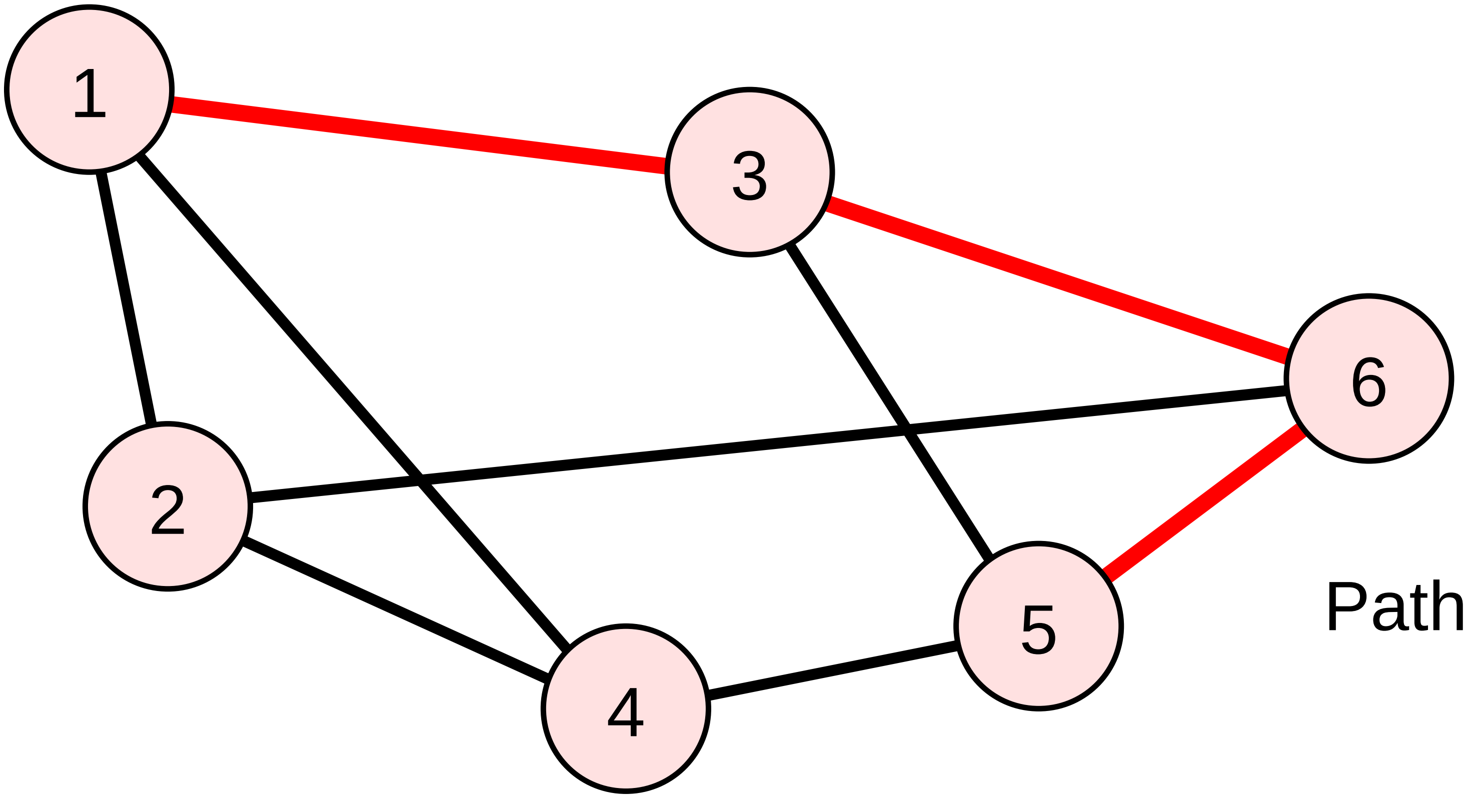
Network Paths



Path 1–3–6–2–4–5

Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5,
1-3-6-2-4-5

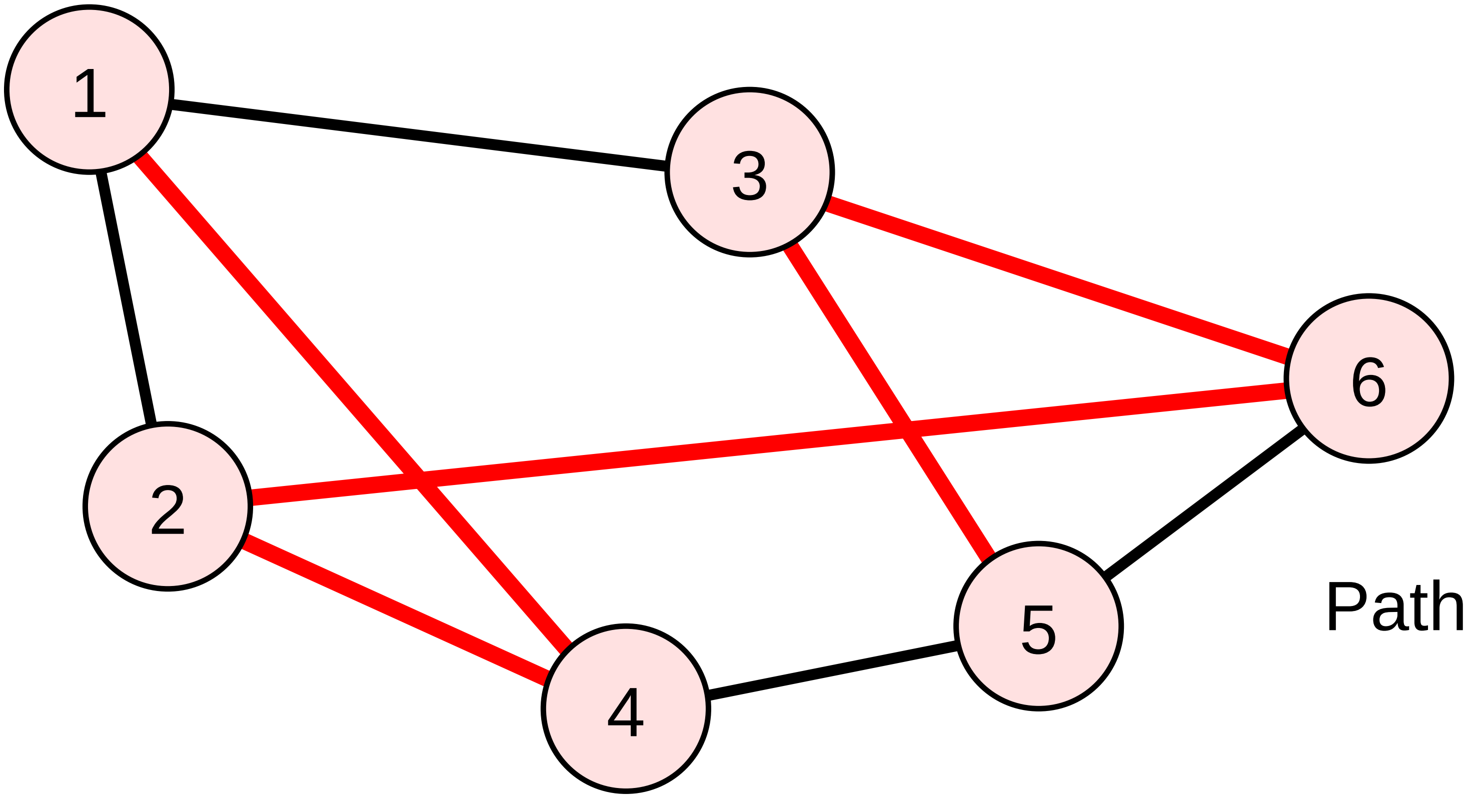
Network Paths



Path 1–3–6–5

Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5

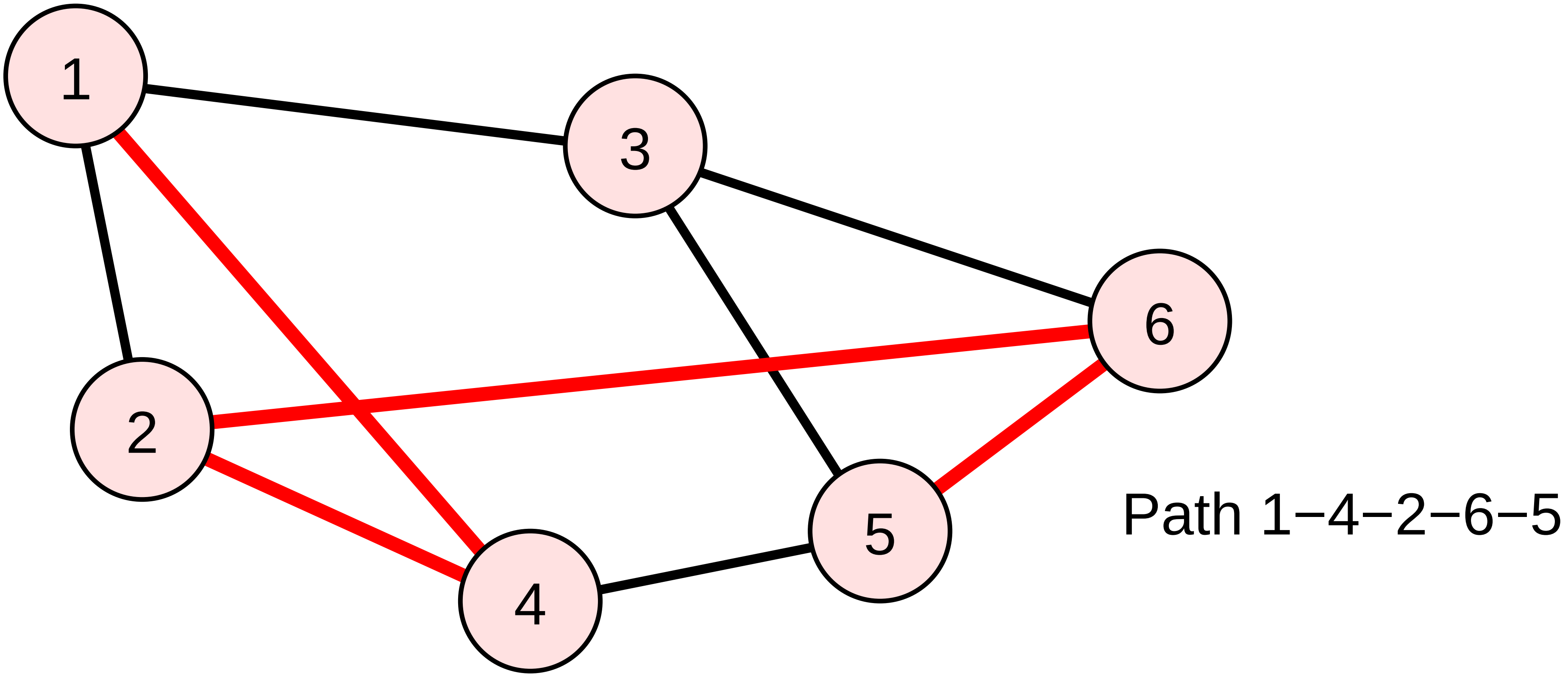
Network Paths



Path 1−4−2−6−3−5

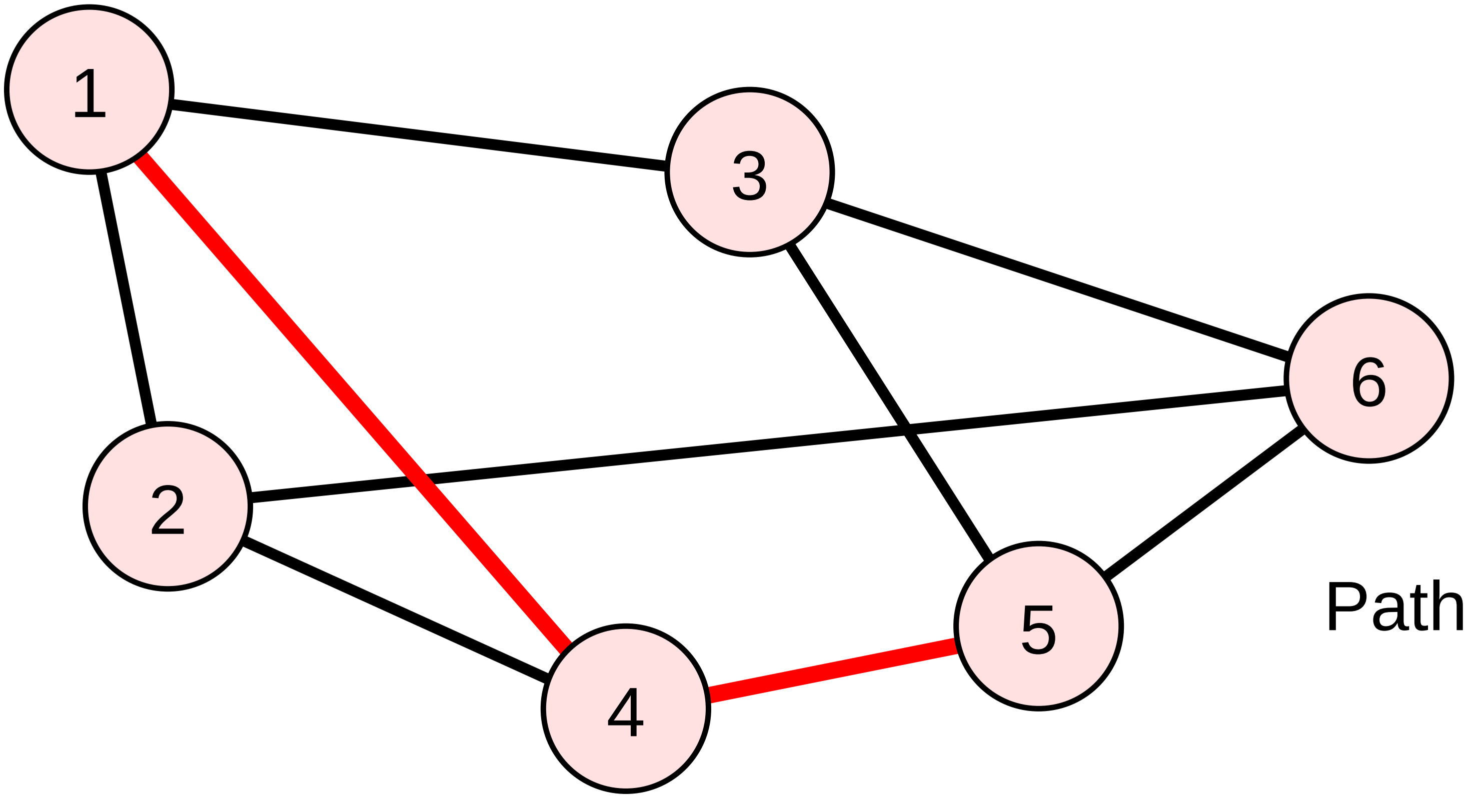
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5, 1-4-2-6-3-5

Network Paths



Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5, 1-4-2-6-3-5, 1-4-2-6-5

Network Paths



Path 1−4−5

Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5,
1-3-6-2-4-5, 1-3-6-5, 1-4-2-6-3-5, 1-4-2-6-5, 1-4-5

Notation and Assumptions

- Each link $e \in E$ has a **capacity**, denoted by $r_e (\geq 0)$.
  - In communication networks, this is the maximum service rate, with units of bits/sec (“bit rate”).
  - If links are uncapacitated,

$$r_e = \begin{cases} \infty, & \forall\, e \in E \\ 0, & \forall\, e \notin E \end{cases}$$

- Links have a physical **distance**, often measured in terms of propagation delays $d_e (\geq 0)$.
  - Where required, assume $d_e = \infty$, $\forall e \notin E$

Routing

- in essence, routing maps
  - end-to-end traffic from p to q, i.e. t_{pq}
  - to end-to-end paths in P_{pq}
  - to links in E
- there are very many paths
  - can't search them all
  - have to be clever about choice of paths
- can use multiple paths
  - load-balancing — spreads load over paths

Routing

Want to route traffic t_{pq} from node p to q

Decision variables are x_μ

$$x_\mu = \text{traffic allocated to path } \mu \in P.$$

Note that $x_\mu \geq 0$ and for all $[p,q] \in K$ and

$$\sum_{\mu \in P_{pq}} x_\mu = t_{pq}$$

Also the x_μ are disjoint

- traffic routed on path $\mu \in P_{pq}$ comes from only t_{pq}.

The vector $\mathbf{x} = (x_\mu : \mu \in P)$ is called the **routing**.

Routing costs

Any routing induces **loads** on a link

- Denote the load on link $e \in E$ by f_e.
    - In directed networks, load is called flow
- link loads are obtained by summing the traffic allocated to all paths containing the link e.

$$f_e = \sum_{\mu \in P : e \in \mu} x_\mu$$

- The vector $\mathbf{f} = (f_e : e \in E)$ is called the **load** on the network.

Routing costs

Assume that load induces cost

- loads cause congestion
  - increases delays
  - can be seen as a type of cost
- we may purchase network capacity from a provider
  - they may charge based on usage
- as network grows
  - we add capacity
  - if more load on links, we need to add capacity sooner, which costs us more
- The **cost** of the network for a given load $\mathbf{f}$ is $C(\mathbf{f})$

Routing problem

The Routing Problem: Determine the optimal routing $\mathbf{x}$ to minimise $C(\mathbf{f})$

Formulation: minimize $C(\mathbf{f})$ s.t.

$$\begin{aligned} f_e &= \sum_{\mu \in P : e \in \mu} x_\mu, && \forall e \in E \\ x_\mu &\geq 0, && \forall \mu \in P \\ \sum_{\mu \in P_{pq}} x_\mu &= t_{pq}, && \forall [p,q] \in K \\ f_e &\leq r_e, && \forall e \in E \end{aligned}$$

Routing problem

The Routing Problem: Determine the optimal routing $\mathbf{x}$ to minimise $C(\mathbf{f})$

Formulation: minimize $C(\mathbf{f})$ s.t.

$$
\begin{aligned}
f_e &= \sum_{\mu \in P : e \in \mu} x_\mu, && \forall e \in E \\
x_\mu &\geq 0, && \forall \mu \in P \\
\sum_{\mu \in P_{pq}} x_\mu &= t_{pq}, && \forall [p,q] \in K \\
f_e &\leq r_e, && \forall e \in E
\end{aligned}
$$

edges

nduced loads

paths

routing

traffic conservation

O-D pairs

capacity constraints

Linear costs

- Remove capacity constraints
- Assume linear costs, with generic weights α_e

$$C(\mathbf{f}) = \sum_{e \in E} \alpha_e f_e, \qquad \alpha_e \geq 0, \forall e \in E$$

- then the cost of using the link is directly proportional to the load on the link, i.e.

$$C(f_e) \propto f_e$$

- α_e is sometimes called
  - the **length** of the link
  - the link ***weight***
  - the link **cost**

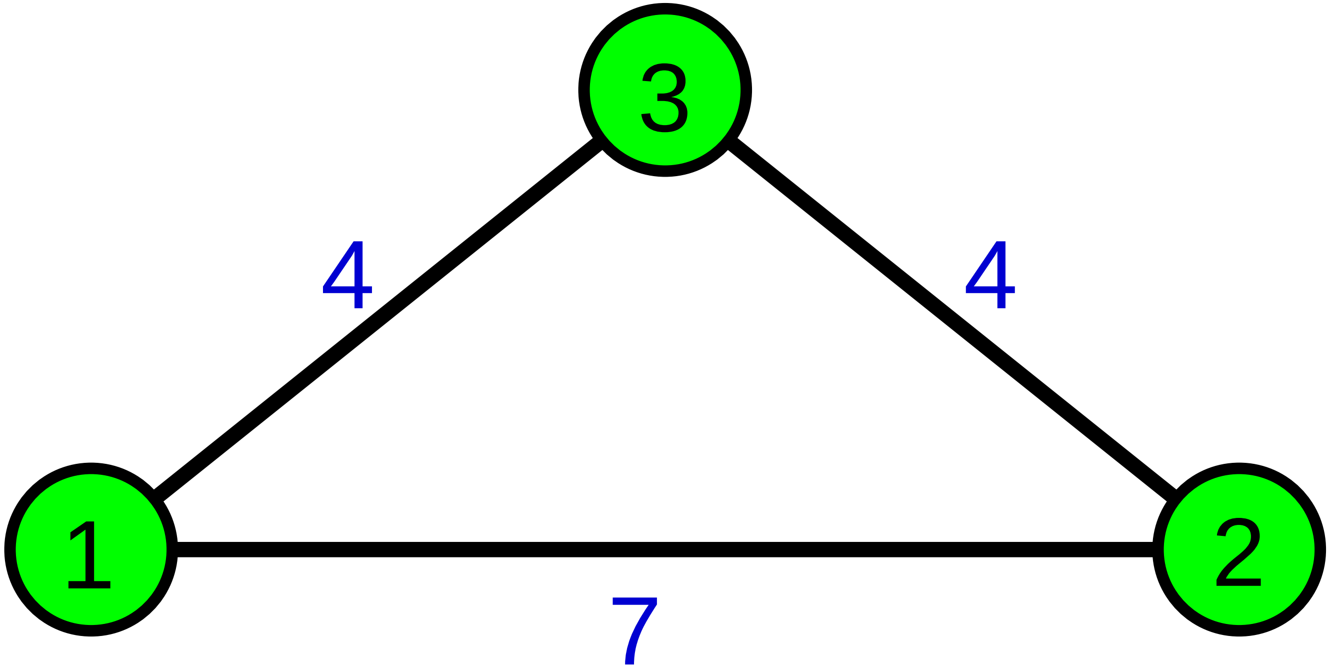
Path lengths

Then, in terms of the decision variables,

$$\begin{aligned} C(\mathbf{f}) &= \sum_{e \in E} \alpha_e f_e \\ &= \sum_{e \in E} \alpha_e \left(\sum_{\mu \in P : e \in \mu} x_\mu \right) \\ &= \sum_{\mu \in P} \left(\sum_{e \in \mu} \alpha_e \right) x_\mu \\ &= \sum_{\mu \in P} l_\mu x_\mu \end{aligned}$$

- $l_\mu = \sum_{e \in \mu} \alpha_e$ is called the **cost**, or **length** of path $\mu \in P$.
- It is the sum of all the link costs along the path
- Relationship between link cost, and path length
    - longer paths use more resources

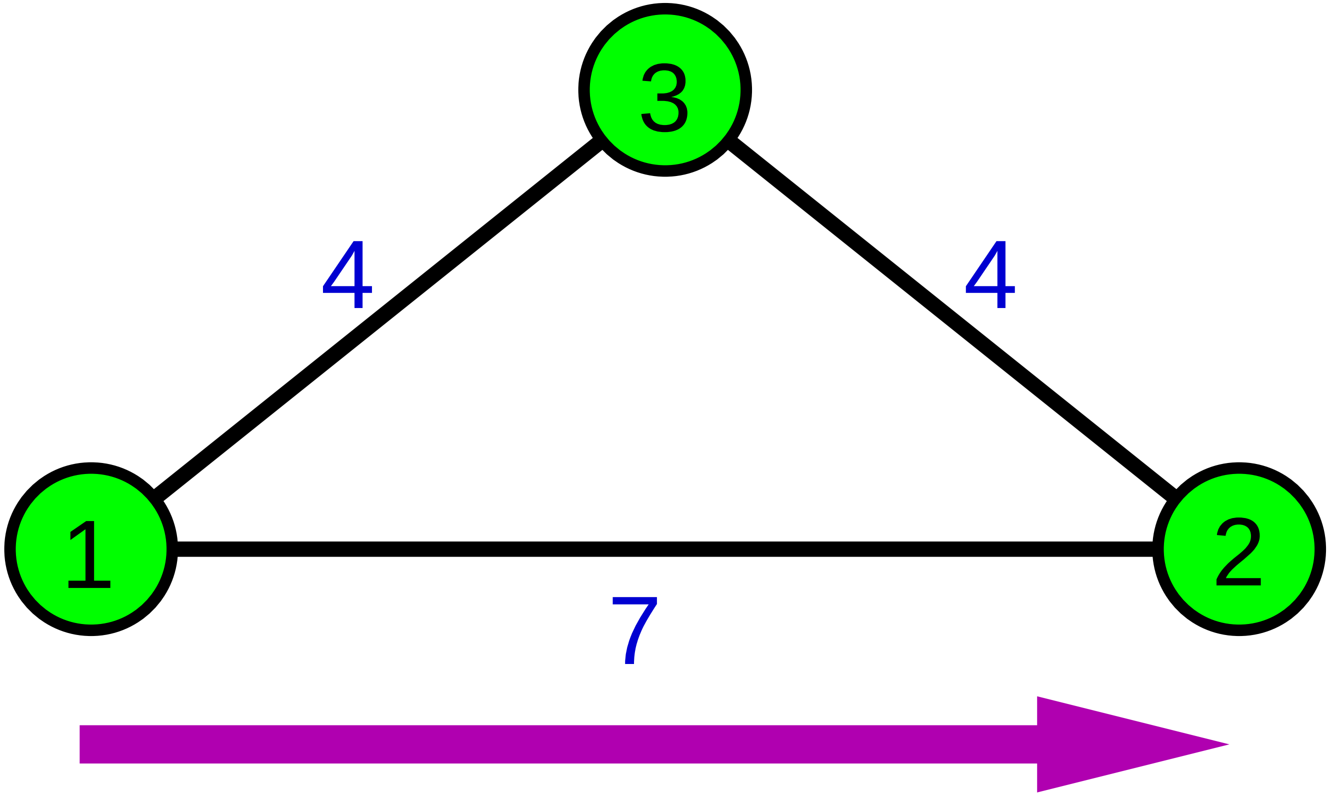
Network path-length example



Two possible paths from 1 -> 2

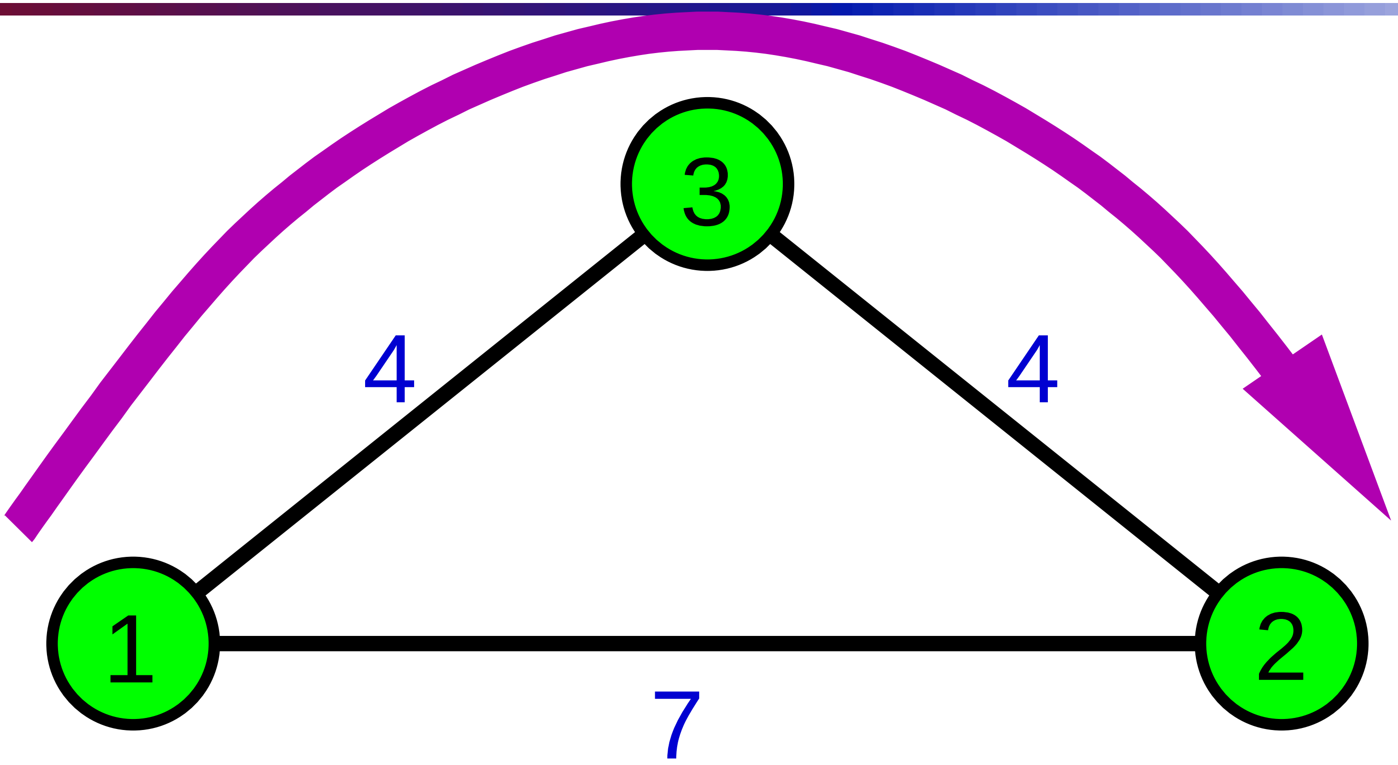
- Path 1 (1-2), and has length $l_\mu = 7$
- Path 2 (1-3-2), and has length $l_\mu = 4 + 4 = 8$

Network path-length example



Shortest path routing

Network path-length example



Longer path uses two links, and hence more resources

Linear costs => shortest path routing

We want to minimize $C(\mathbf{f}) = \sum_{e \in E} \alpha_e f_e = \sum_{\mu \in P} l_\mu x_\mu$

- find minimum length paths $\hat{l}_{pq} = \min\{l_\mu : \mu \in P_{pq}\}$
- put all traffic t_{pq} on a minimum length path
- then we get cost

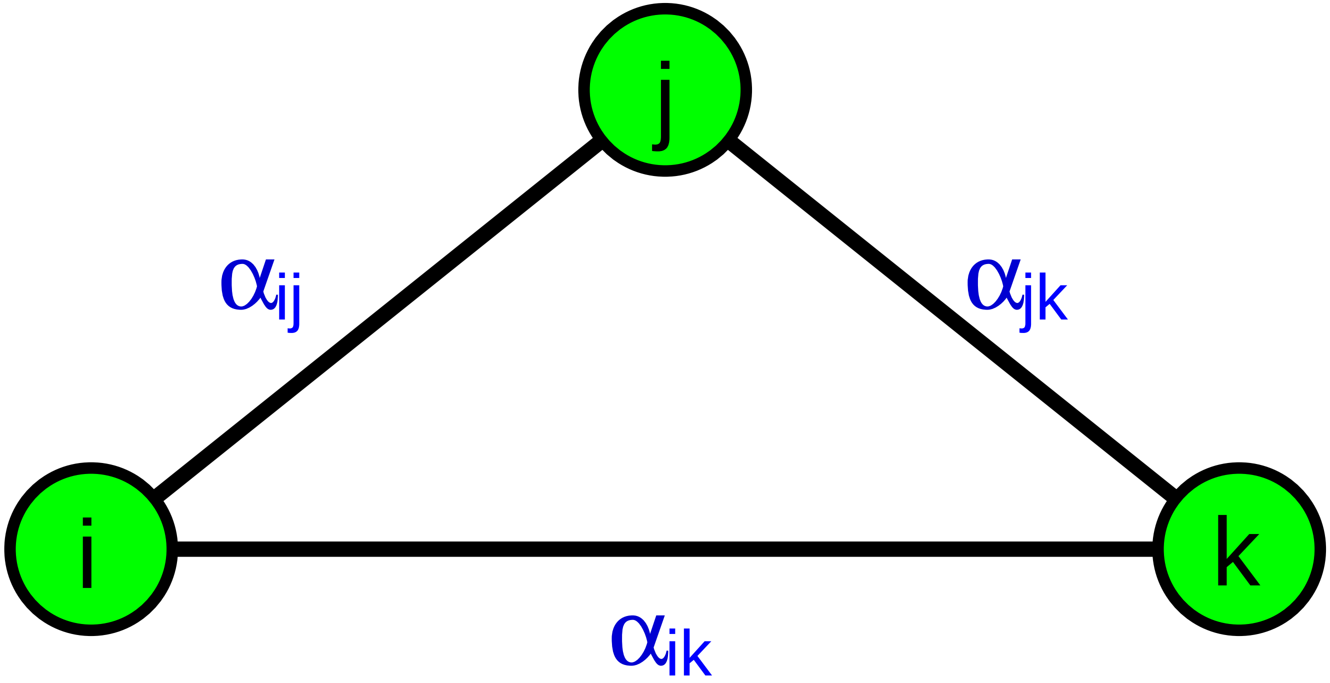
$$C(\mathbf{f}) = \sum_{\mu \in P} l_\mu x_\mu = \sum_{[p,q] \in K} \hat{l}_{pq} t_{pq}$$

- problem solved!
  - we just have to find shortest paths
    - Dijkstra's algorithm
    - Floyd-Warshall algorithm

Special case

- the network is fully meshed (a clique),
  $$E = \{(i,j),\ \forall i,j \in N, i \neq j\}$$
- the α_e satisfy the triangle inequality i.e.
  $$\alpha_{ik} \leq \alpha_{ij} + \alpha_{jk}, \quad \forall\ i,k,j \in N$$
- Then the path of minimum cost between any two nodes p,q is the direct link (p,q).
- That is, we route all offered traffic t_{pq} directly from p to q.
- This network is called:
  a fully meshed network (or clique) with direct link routing.

Triangle inequality



$$\alpha_{ik} \leq \alpha_{ij} + \alpha_{jk}, \quad \forall\ i,k,j \in N$$

Dijkstra's algorithm

- most networks are not cliques
- fast method to find shortest paths is **Dijkstra's algorithm** [1]
    - Edsger Dijkstra (1930-2002)
        - Dutch computer scientist
        - Turing prize winner 1972.
        - "Goto Statement Considered Harmful" paper
- find distance of all nodes from one start point
- works by finding paths in order of shortest first
    - longer paths are built up of shorter paths

Dijkstra's algorithm

Input

- graph (N,E)
- link **weights** α_e, define link distances

$$d_{ij} = \begin{cases} 0 & \text{if } i = j \\ \alpha_e & \text{where } (i,j) = e \in E \\ \infty & \text{where } (i,j) = e \notin E \end{cases}$$

- a start node, WLOG assume it is node 1

Output

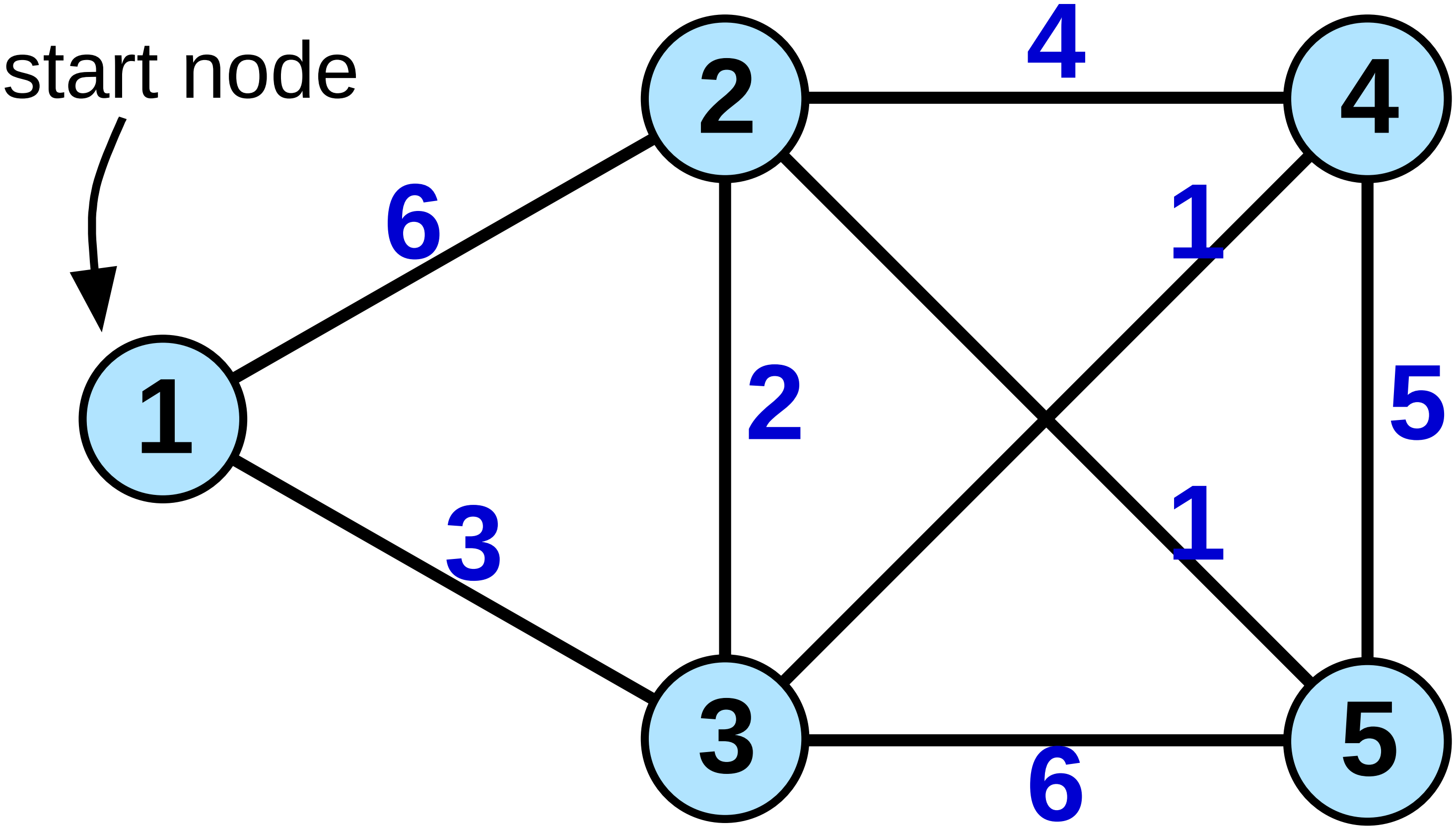
- distances D_j of each node $j \in N$ from start node 1.
- a predecessor node for each node (gives path)

Dijkstra's algorithm

Let S be the set of labelled nodes.

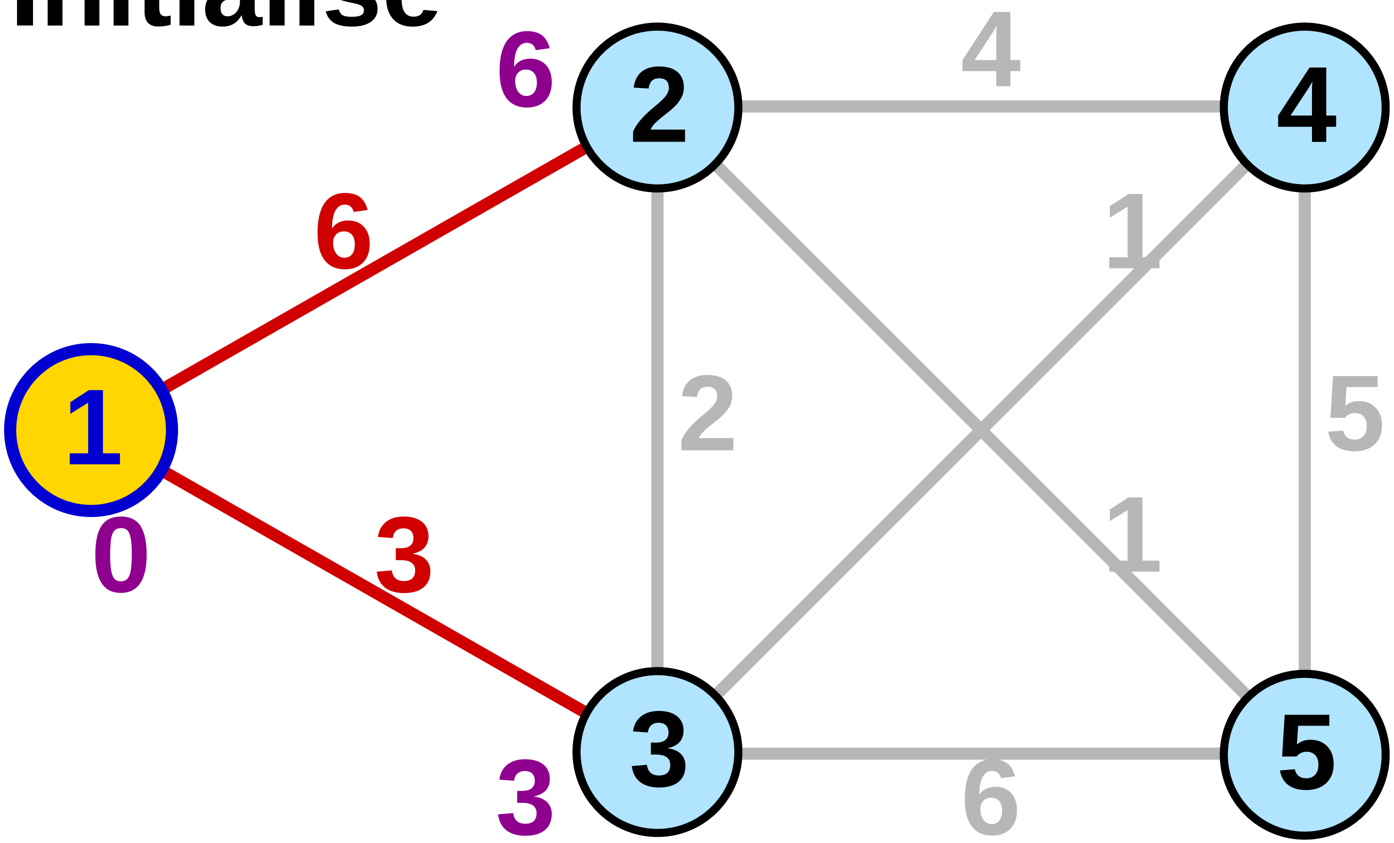
```
Initialise: $S = \{1\}$,
            $D_1 = 0$,
            $D_j = d_{1j}$, $\forall j \notin S$, i.e. $j \neq 1$.

Step 1: Find the next closest node
            Find $i \notin S$ such that $D_i = \min\{D_j : j \notin S\}$
            Set $S = S \cup \{i\}$.
            If $S = N$, **stop**

Step 2: Find new distances
            For all $j \notin S$, set
            $D_j = \min\{D_j, D_i + d_{ij}\}$
            **Goto Step 1.**
```

Dijkstra Example



Dijkstra Example

Initialise

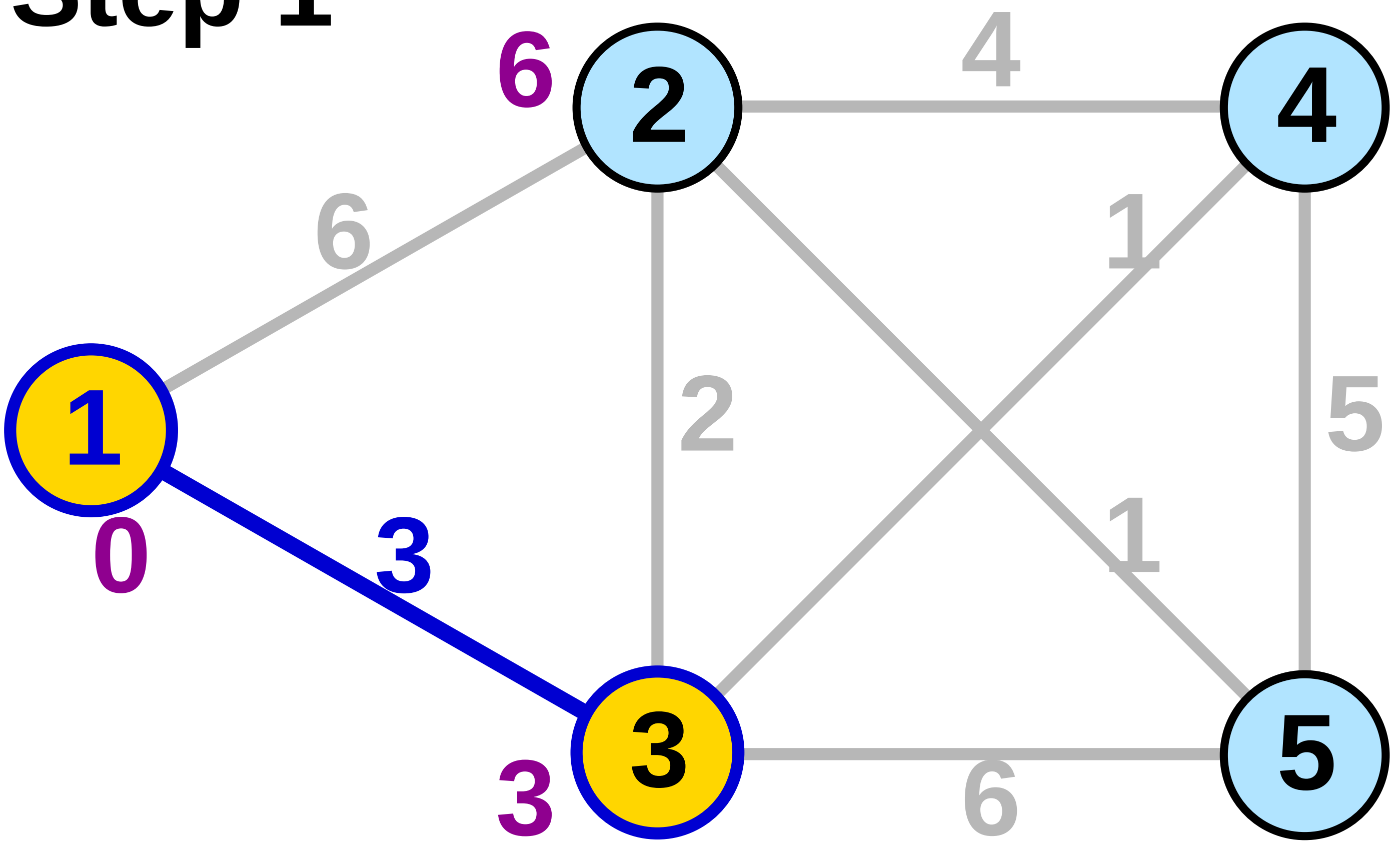


S={1}

D=(0,6,3, ,)

Dijkstra Example

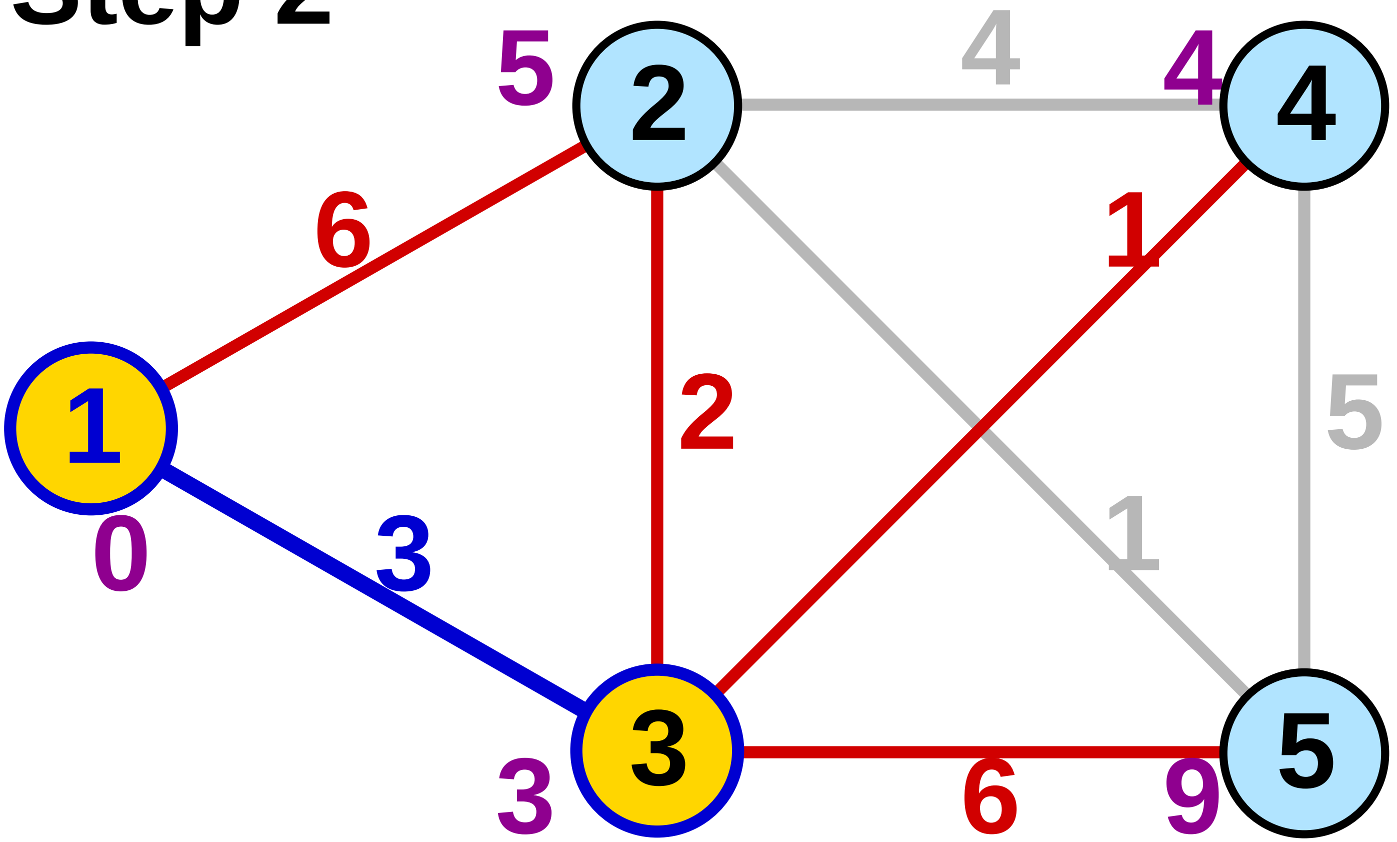
Step 1



S={1,3}   D=(0,6,3, ,)

Dijkstra Example

Step 2

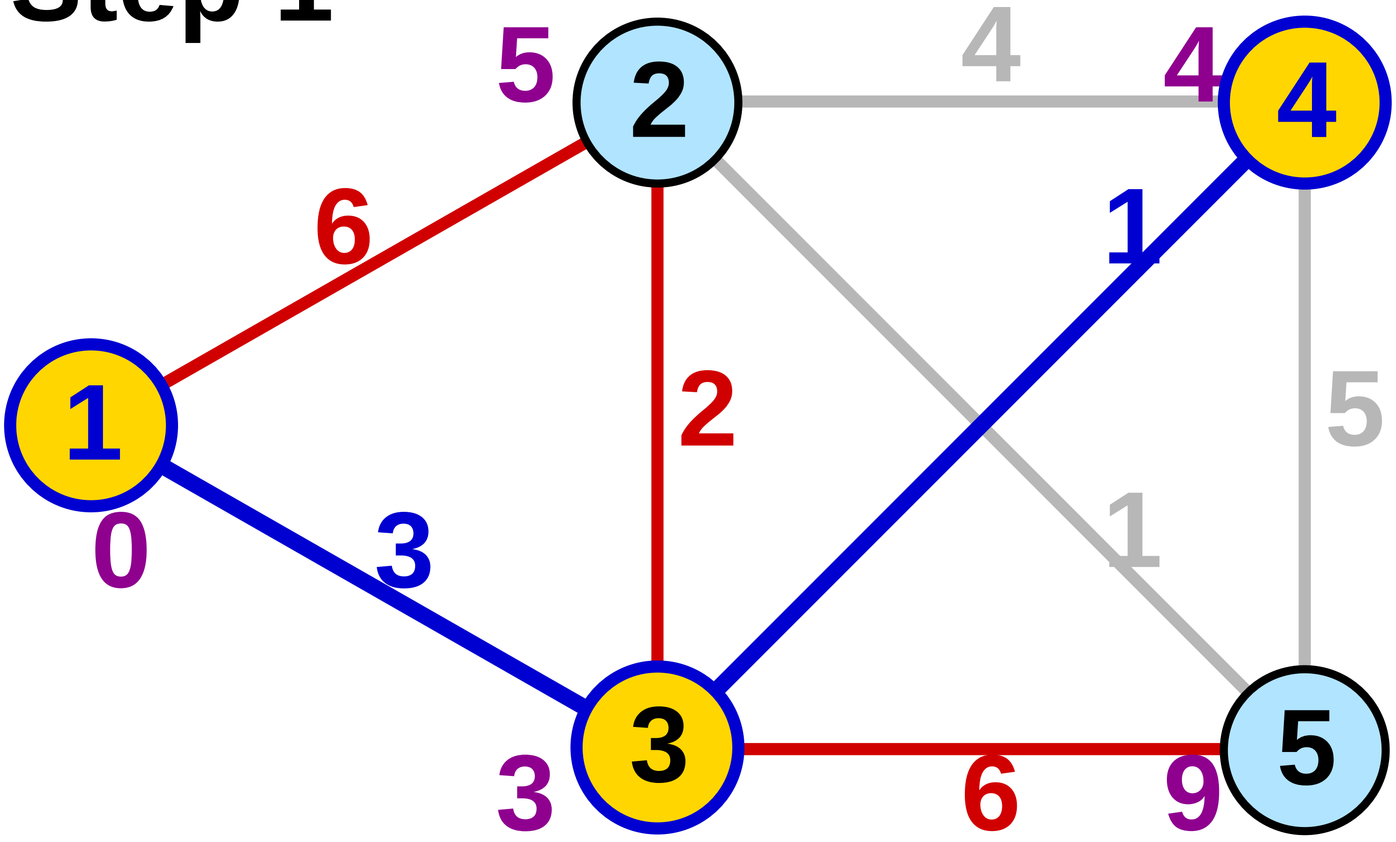


S={1,3}

D=(0,5,3,4,9)

Dijkstra Example

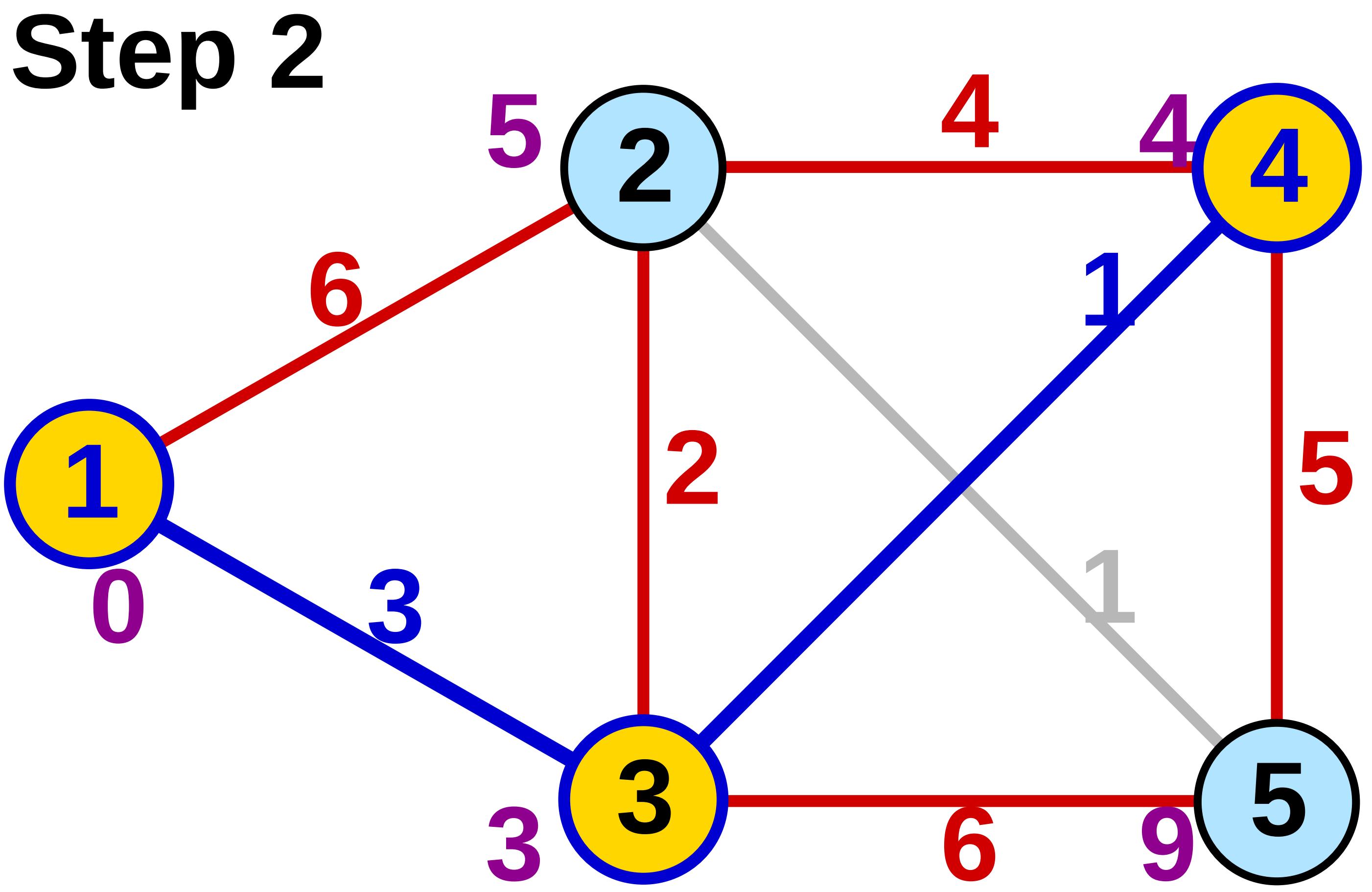
Step 1



S={1,3,4}

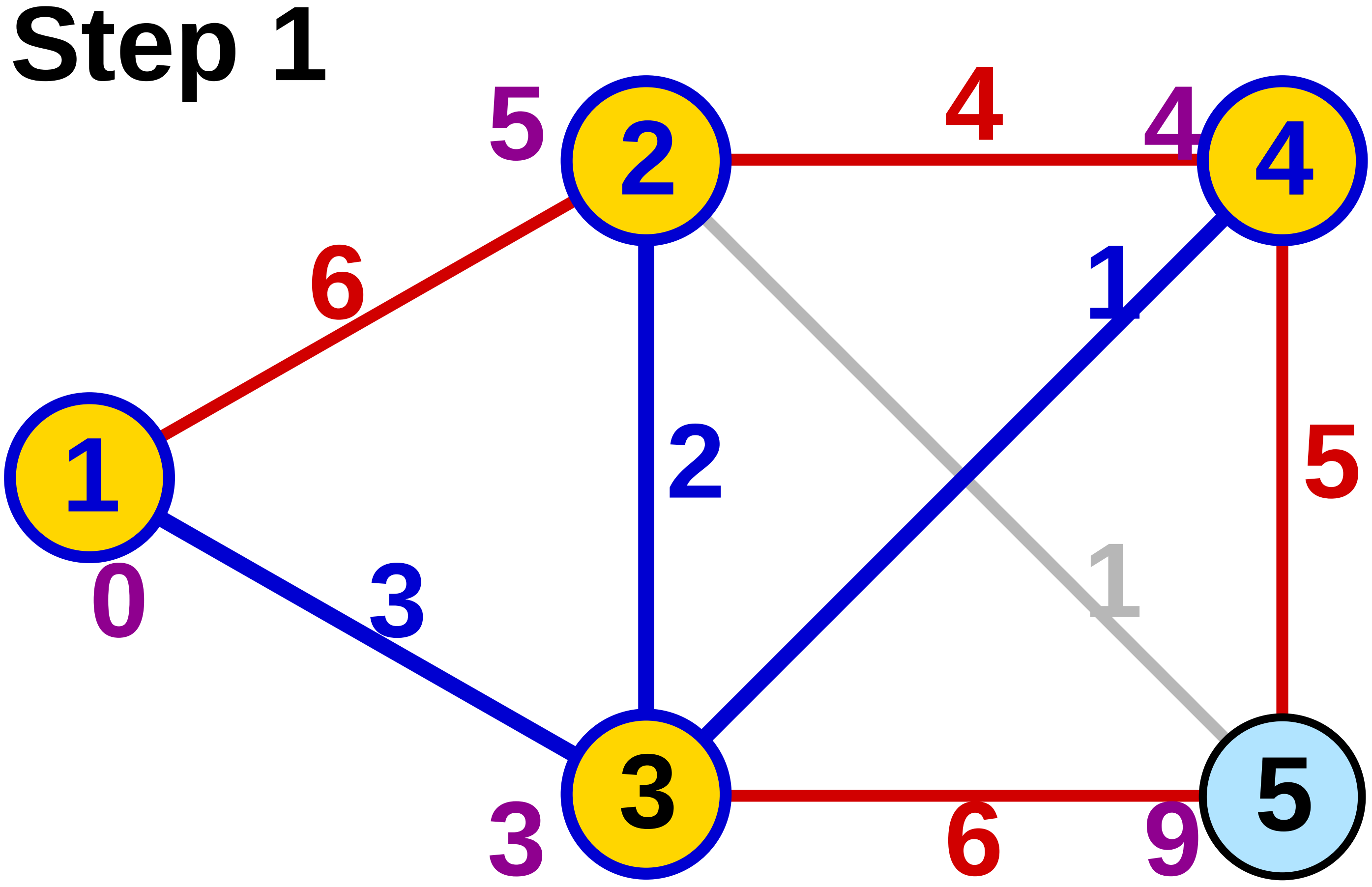
D=(0,5,3,4,9)

Dijkstra Example



S={1,3,4}   D=(0,5,3,4,9)

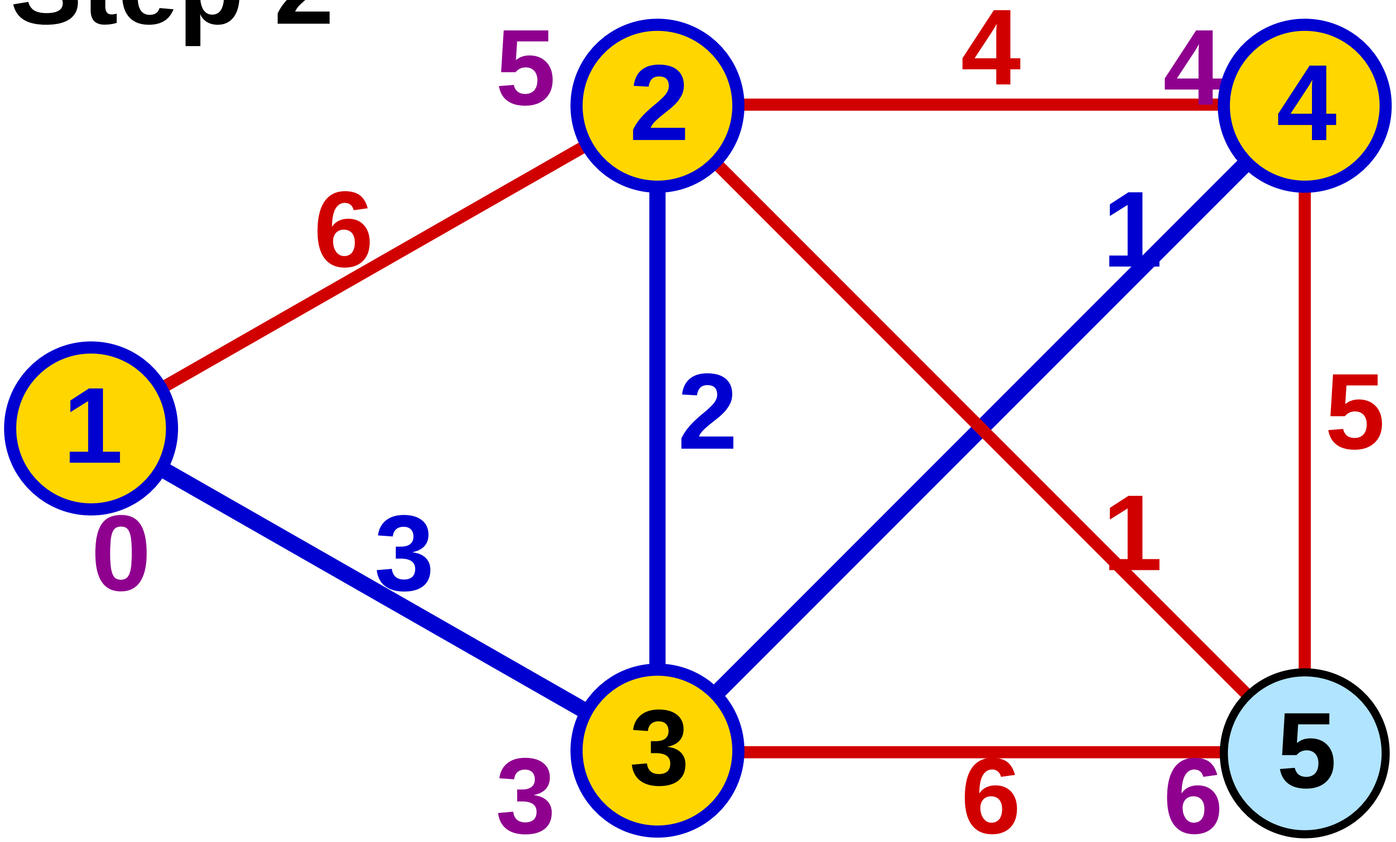
Dijkstra Example



S={1,3,4,2}   D=(0,5,3,4,9)

Dijkstra Example

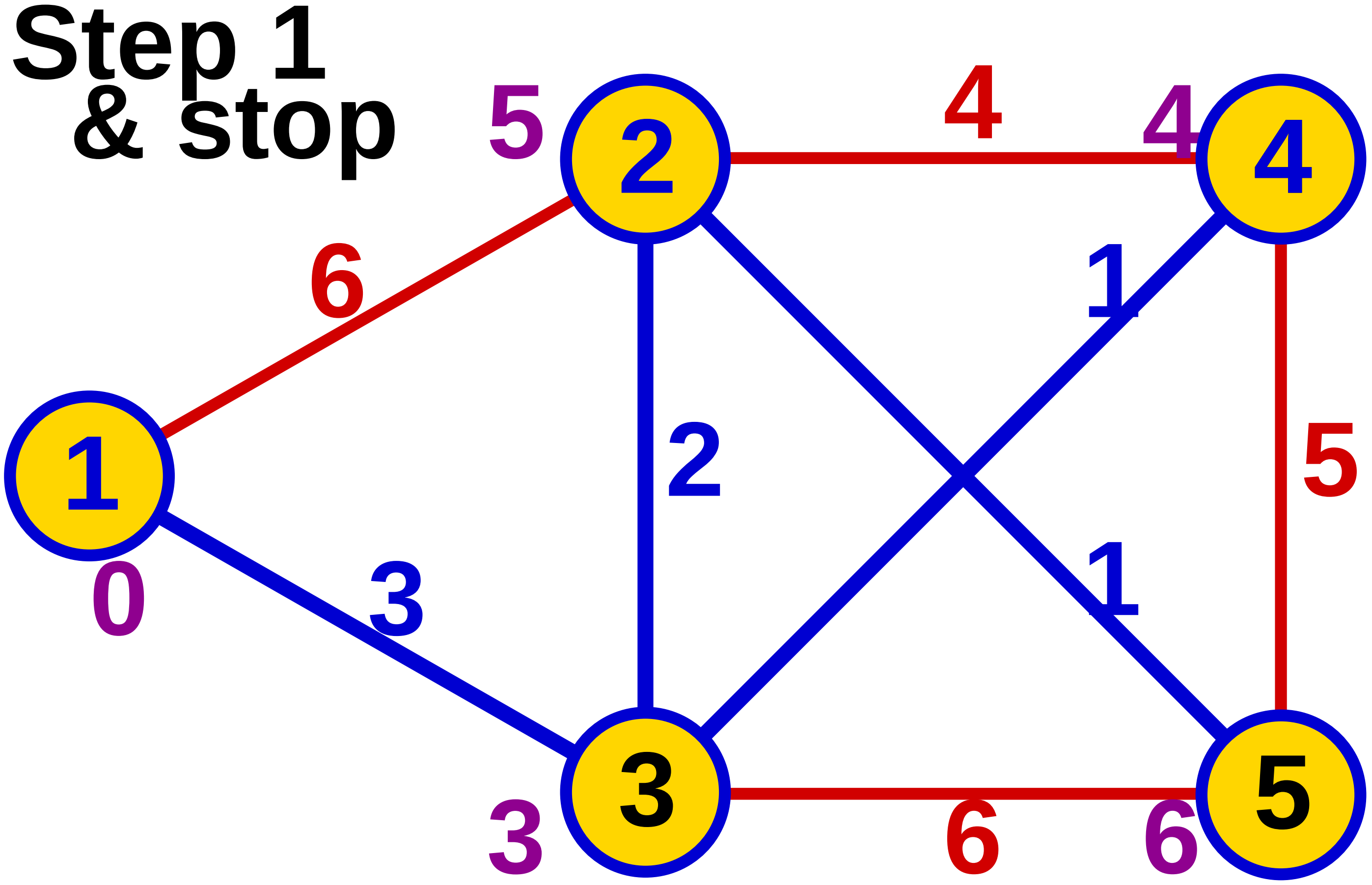
Step 2



S={1,3,4,2} **D=(0,5,3,4,6)**

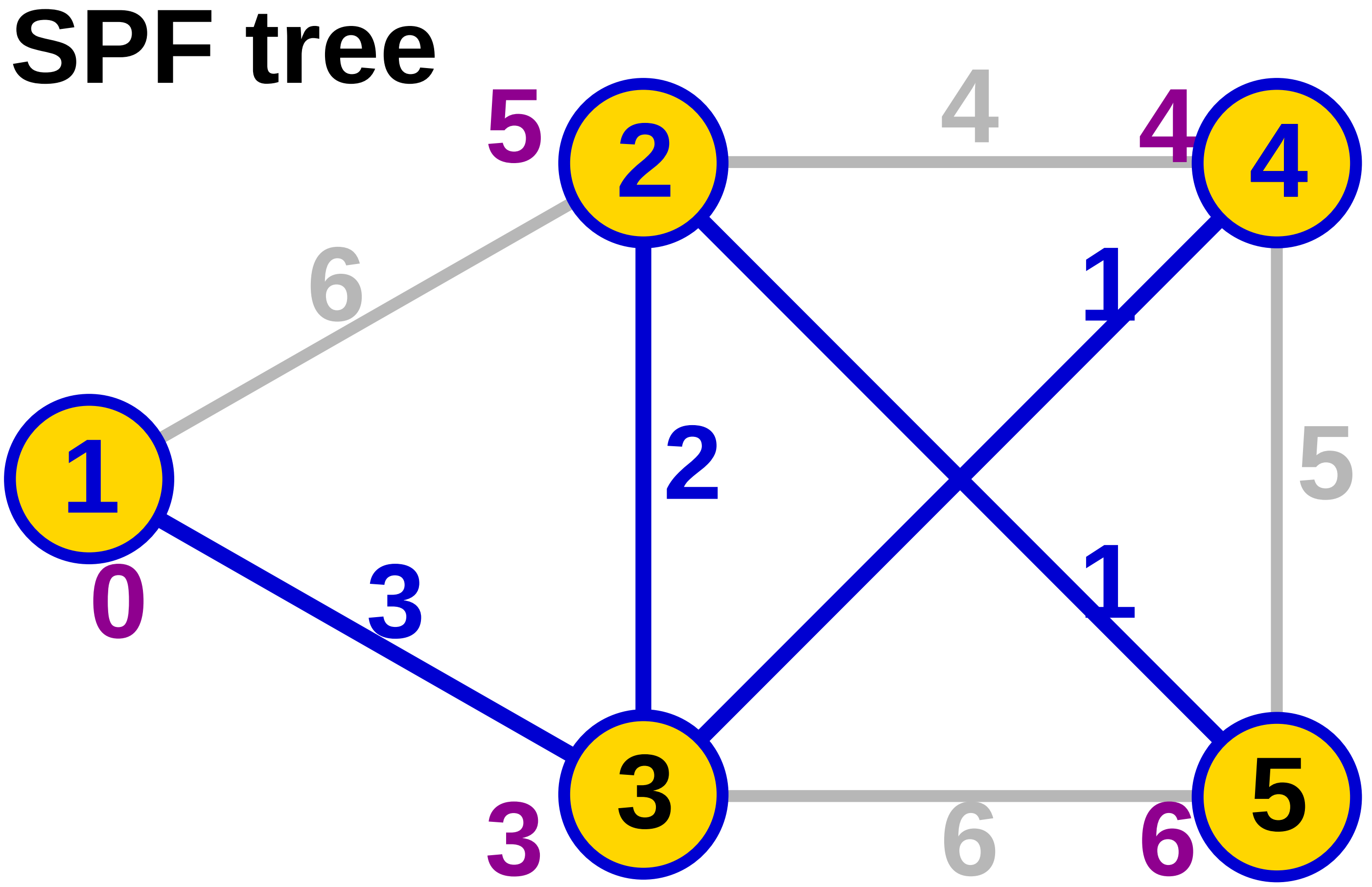
changed

Dijkstra Example



S={1,3,4,2,5} D=(0,5,3,4,6)

Dijkstra Result



S={1,3,4,2,5}   D=(0,5,3,4,6)

Dijkstra intuition

- build a (Shortest-Path First) **SPF tree**
- let it grow
- grow by adding shortest paths onto it
- solution must look like a tree
    - to get paths, we only need to keep track of **predecessors**, e.g. previous example

node	predecessor
1	-
2	3
3	1
4	3
6	2

Dijkstra issues

- Dijkstra's algorithm solves **single-source all-destinations problem**
- easily extended to a directed graph
  - can only join up in the direction of a link
- link-distances (weights) must be non-negative
  - there are generalizations to deal with negative weights
  - not often needed for communications networks

For more examples use

`http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html`

Dijkstra complexity

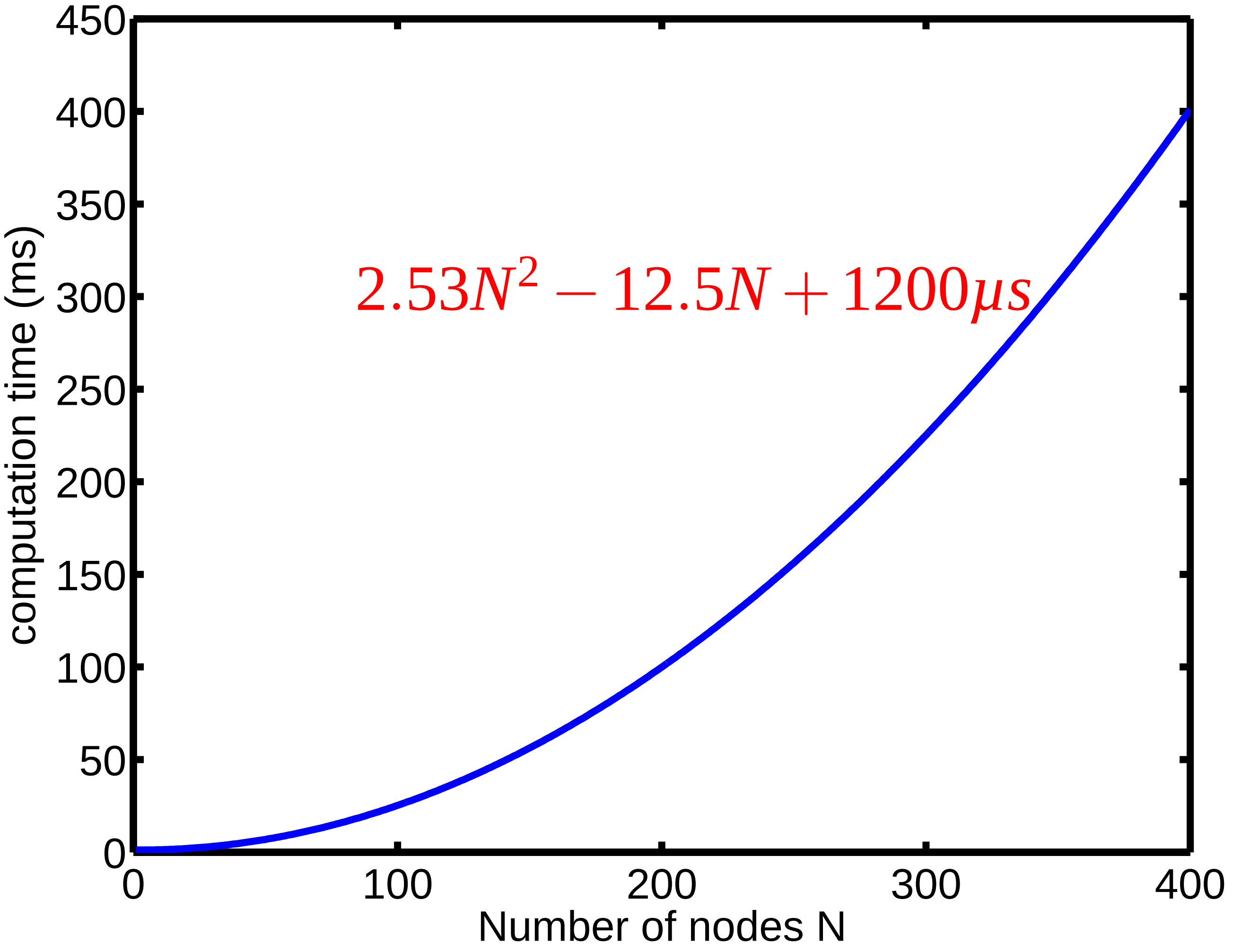
- simple implementation complexity $O(|N|^2)$
- Cisco's implementation of Dijkstra tested in [2]

$$\text{comp.time} = 2.53N^2 - 12.5N + 1200 \text{ microseconds}$$

- complexity (assuming smart data structures, i.e. Fibonacci heap) is $O(|E| + |N| \log |N|)$,
  - $|E|$ = number of edges
  - $|N|$ = number of nodes
- to compute paths for all pairs, we can perform Dijkstra for each starting point, with complexity $O(|N||E| + |N|^2 \log |N|)$,

Dijkstra complexity

Empirical Cisco 7500 and 12000 (GSR) computation times for Dijkstra [2]



Sketch of proof of Dijkstra

Dijkstra's algorithm solves the single-source shortest-paths problem in networks that have nonnegative weights.

Proof: Call the source node s the root, then we need to show that the paths from s to each node x corresponds to a shortest path in the graph from s to x. Note that this set of paths forms a tree out of a subset of edges of the graph.

The proof uses induction. We assume that the subtree formed at some point along the algorithm has the property (of shortest paths). Clearly the starting point satisfies this assumption, so we need only prove that adding a new node x adds a shortest path to that node. All other paths to x must begin with a path from the current subtree (because these are shortest paths) followed by an edge to a node not on the tree. By construction, all such paths are longer than the one from s to x that is produced by Dijkstra.

References

[1] E. Dijkstra, "A note in two problems in connexion with graphs," Numerische Mathematik, vol. 1, pp. 269–271, 1959.

[2] A. Shaikh and A. Greenberg, "Experience in black-box OSPF measurement," in Proc. ACM SIGCOMM Internet Measurement Workshop, pp. 113–125, 2001.