# Routing in Switched Networks

Dr Steve Gordon

ICT, SIIT

# Contents

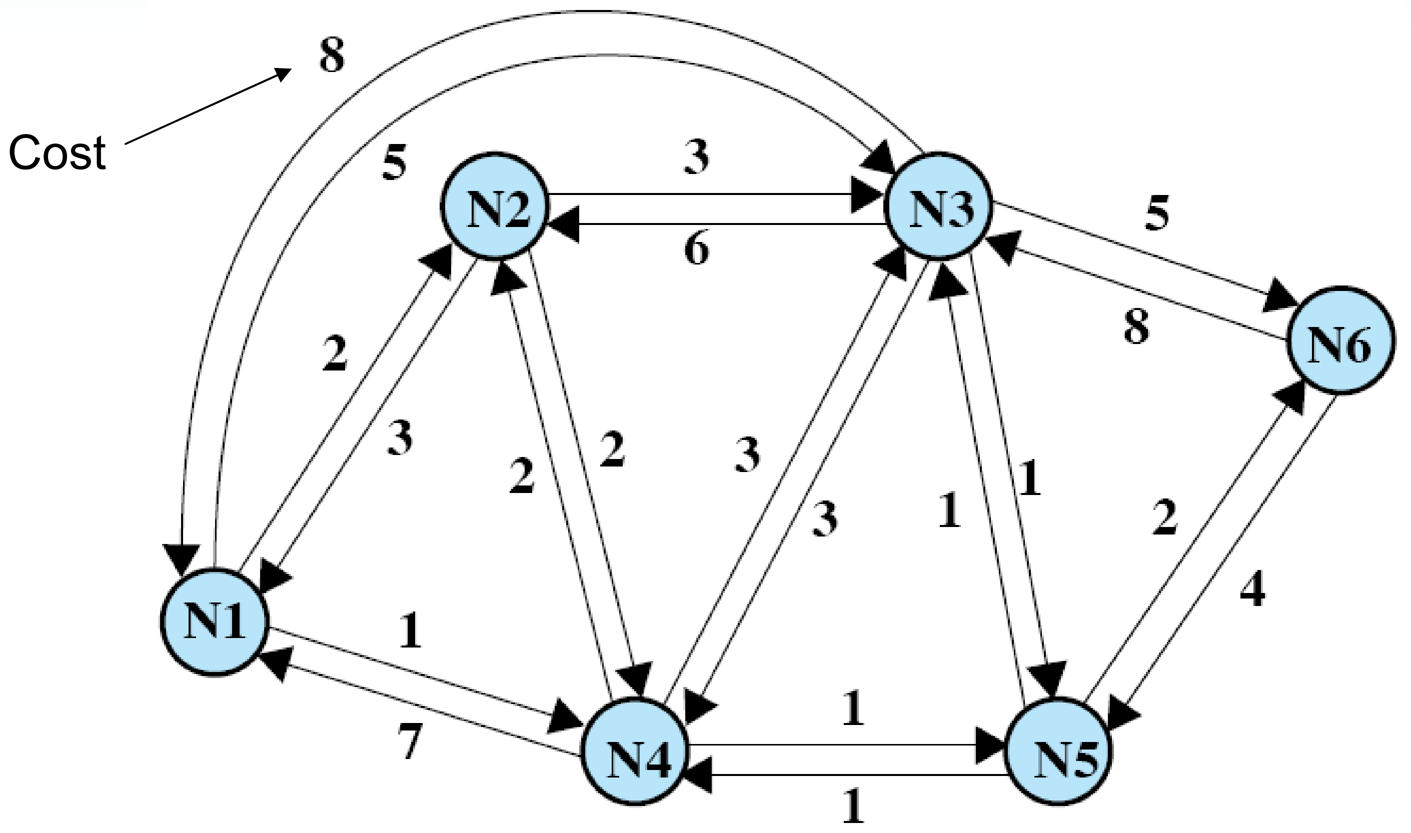
# Routing in Switched Networks

- Routing is a key design issue in switched networks
  - What path (route) should be taken from source to destination?
    - Generally, more than one path is possible
    - Want to choose the “best” path
  - Applicable to circuit-switched and packet-switched networks
- We will look at general routing strategies
  - Focus on packet-switched networks
  - Later topics will look at Internet routing in detail

## Characteristics of Routing Functions

- A routing function (algorithm) must meet many requirements:
  - Correctness
    - Must choose a path from intended source to intended destination
  - Simplicity
    - Easy and cheap to implement
  - Robustness
    - In presence of errors or overload in some parts of network, network should react to reduce load and find alternate paths
  - Stability
    - New paths (e.g. to avoid overload) should not be too frequent
  - Fairness
    - Some algorithms may give higher priority to stations close together (shorter paths) than further away (longer paths) – maybe efficient, but unfair
  - Optimality
    - Choose best paths
  - Efficiency
    - Minimise the amount of processing and transmission overhead

# Example Network Configuration



Link: a direct connection between two nodes (such as via cable or wireless). E.g. link fro N1 to N2
Path (or route): a track or way between two nodes, via one or more links. E.g. a path from N1 to N6 is N1 – N2 – N3 – N6
Hop: traverse a link. E.g. there are 3 hops between N1 and N6 on path N1 – N2 – N3 – N6

# Elements of Routing Techniques

- Select a route based on Performance Criteria
  - Least-cost routing
    - Cost can be different criteria: number of hops, delay, throughput, financial, …
    - Costs are assigned to links, and routing selects a path with least cost
- When and where do you make the routing decision?
  - Time
    - For each packet
    - For each virtual circuit
  - Place (which node makes the decision)
    - Distributed: each node selects an output link for incoming packets (robust, but complex)
    - Centralised: central node decides
    - Source routing: source station decides

## Elements of Routing Techniques

- Network information source and updating
  - Often nodes need to know about topology, link costs and load to make routing decisions
    - How do nodes collect this information and how often is it updated?
      - More frequent updates give more reliable information and hence better routing decisions
        - But increases overhead (consumes network resources)
  - Source of information may be ("where does the information come from?")
    - None: no information about the network is used
    - Local: use the information only known to the current node
    - Adjacent node: your neighbour nodes send you information
    - Nodes along route: the nodes along the path/route send you information
    - All nodes: all nodes send you information
  - Updating may be ("when does the information come?"):
    - Never: if no information about the network is used, you never update!
    - Continuous: if using local information, essentially you have access all the time
    - Periodic: at regular intervals
    - Major load change: in increase/decrease of traffic on a link
    - Topology change: a new node is added/removed from network, or links are changed

## Routing Strategies

Fixed
Flooding
Random
Adaptive

# Fixed Routing

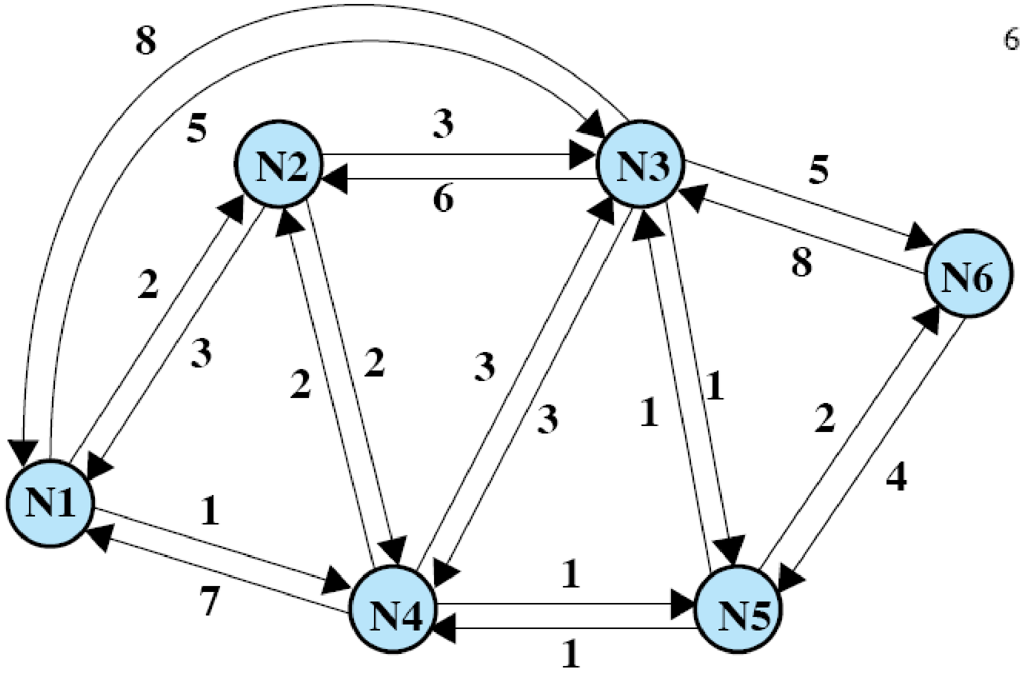
- Use a single permanent route for each source to destination pair
  - Determined using a least cost algorithm
    - The metric can be anything. For example: monetary cost, delay, capacity, hops or a combination
  - Route is fixed
    - At least until a change in network topology
    - Hence cannot respond to traffic changes (e.g. overload in one portion of the network)
  - No difference between routing for datagrams and virtual circuits
  - Advantage is simplicity
    - You assign the routes at the start, and then nothing to do
  - Disadvantage is lack of flexibility
    - When the network is operating, changes such as load or topology may mean better routes than initially selected should be used

# Fixed Routing Example

- A network control centre stores routes between all nodes
  - Shows the next node in route E.g. next node from 1 to 3 is 4
- Routing tables can be developed and stored at individual nodes (next slide)

**CENTRAL ROUTING DIRECTORY**

| To Node ↓ / From Node → | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | — | 1 | 5 | 2 | 4 | 5 |
| 2 | 2 | — | 5 | 2 | 4 | 5 |
| 3 | 4 | 3 | — | 5 | 3 | 5 |
| 4 | 4 | 4 | 5 | — | 4 | 5 |
| 5 | 4 | 4 | 5 | 5 | — | 5 |
| 6 | 4 | 4 | 5 | 5 | 6 | — |



*The routing information can be stored at a central node (as above) or distributed to each node (as the next slide shows)*

# Fixed Routing Example

X

Z

Y

**Node 1 Directory**

| Destination | Next Node |
|---|---|
| 2 | 2 |
| 3 | 4 |
| 4 | 4 |
| 5 | 4 |
| 6 | 4 |

**Node 2 Directory**

| Destination | Next Node |
|---|---|
| 1 | 1 |
| 3 | 3 |
| 4 | 4 |
| 5 | 4 |
| 6 | 4 |

**Node 3 Directory**

| Destination | Next Node |
|---|---|
| 1 | 5 |
| 2 | 5 |
| 4 | 5 |
| 5 | 5 |
| 6 | 5 |

**Node 4 Directory**

| Destination | Next Node |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 5 |
| 5 | 5 |
| 6 | 5 |

**Node 5 Directory**

| Destination | Next Node |
|---|---|
| 1 | 4 |
| 2 | 4 |
| 3 | 3 |
| 4 | 4 |
| 6 | 6 |

**Node 6 Directory**

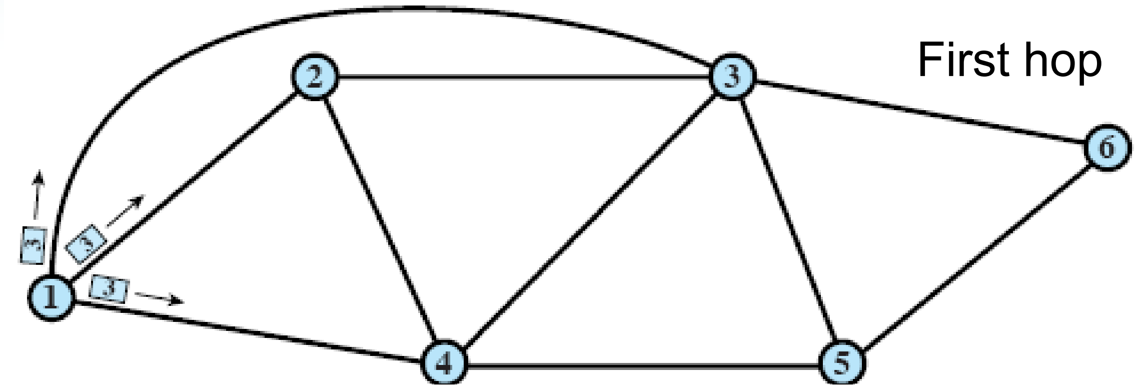
| Destination | Next Node |
|---|---|
| 1 | 5 |
| 2 | 5 |
| 3 | 5 |
| 4 | 5 |
| 5 | 5 |

You can read the tables as: “At Node X, in order to reach Destination Y, send to the Next Node Z”

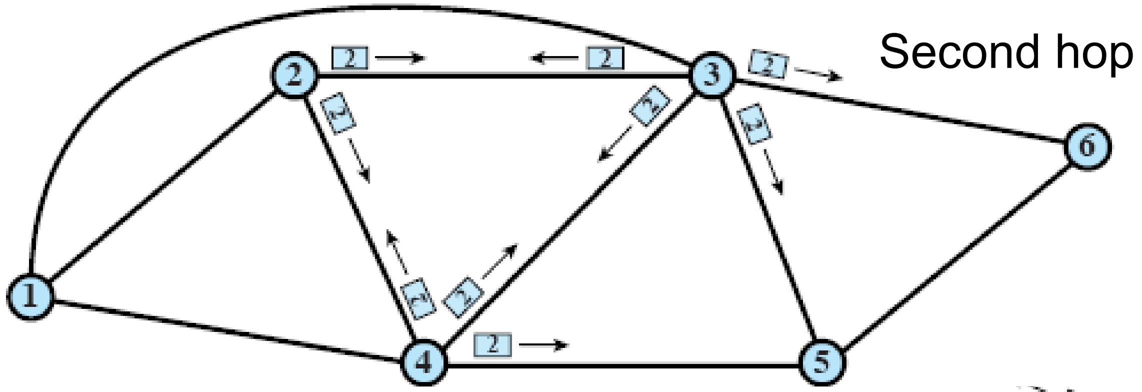
## Flooding

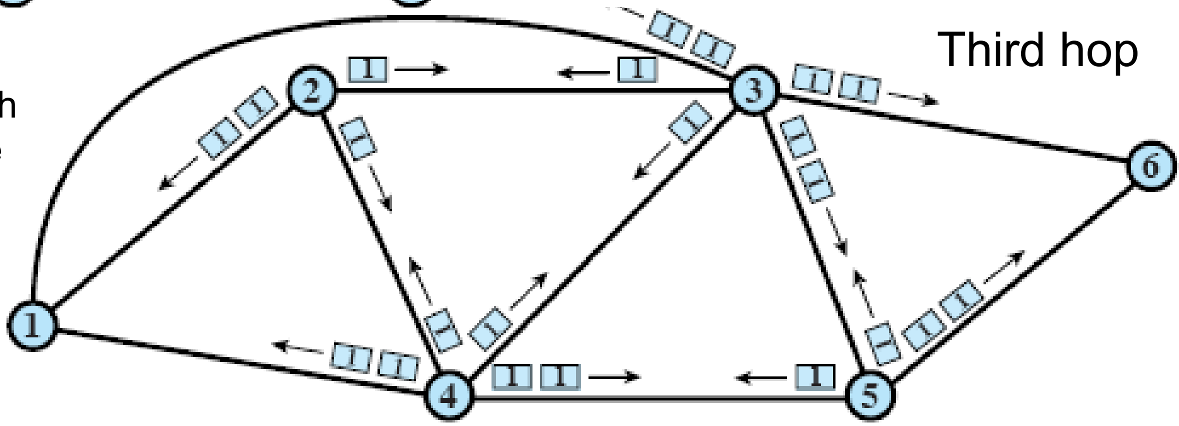
- Packet sent by node to every neighbor
  - Eventually multiple copies arrive at destination
  - No information about the network is required (topology, load)
  - Each packet is uniquely numbered so duplicates can be discarded
  - Need some way to limit continued retransmission
    - nodes can remember packets already forwarded to keep network load in bounds
    - or include a hop count in packets
      - Starts with an initial value (e.g. 5), and every time the packet is forwarded, the value is reduced by 1. If the hop count reaches 0, the packet is not forwarded
- Properties of flooding
  - All possible routes are tried (very robust)
  - At least one packet will have taken minimum hop count route (can be used to set up virtual circuit, as will find the minimum hop path)
  - All nodes are visited (useful to distribute topology information)
  - Disadvantage is high traffic load generated

# Flooding Example



When a node receives a packet, it sends a copy to all links, except the link it received the packet from. If a node receives two copies of a packet, it will send two copies out. That is, the node doesn't care that it is sending two copies of the *same* original packet. In practice, sequence numbers may be used so this doesn't occur.





In this example, one packet is sent (although multiple copies of that packet are sent). The numbers on the packet show the maximum hop count. It is decremented when it is received by a node. If it is 0 after being received, the node will not forward it.

# Random Routing

- Simplicity of flooding with much less load
  - Node selects one outgoing link for retransmission of incoming packet
  - Selection can be random or round robin
    - Round robin: select link 1, then next time link 2, then next time link 3, and so on until all link selected, then return to link 1
  - A refinement is to select outgoing path based on probability calculation
    - E.g. base the probability on the data rate of link; higher data rate, higher the probability that you will select that link
- No network information is needed
  - Therefore no overhead in distributing updates
- But a random route is typically not least cost
  - E.g. you may choose a route that has many hops
  - Therefore network carries higher than optimal load (but not nearly as high as flooding)

## Adaptive Routing

- Used by almost all packet switching networks, e.g. the Internet
- Routing decisions change as conditions on the network change due to failure or congestion
- Requires information about network, e.g. regular updates on the current load and delay of links and routers
- Disadvantages:
  - Decisions more complex (complex algorithms to selecting the best path)
  - Tradeoff between quality of network information and overhead
    - Reacting too quickly can cause oscillation
      - First select path A for packet 1, then path B packet 2, then path A, and so on: varying delays (jitter) and bandwidth; varying loads
    - Reacting too slowly means information may be irrelevant
      - Select path A based on outdated information, when in fact path B is better
- Advantages:
  - Improved performance
    - Potentially can select the most suited paths
  - Can aid congestion control, because tends to balance load

## Classification of Adaptive Strategies

- Classify based on information sources (where does the information come from?):
  - Local (isolated)
    - Route to outgoing link with the shortest queue
    - Can include bias for each destination so that packet goes in general direction
    - Rarely used - does not make use of available information
  - Adjacent nodes
    - Takes advantage on delay and/or outage information, which is exchanged with adjacent nodes
    - Use a least-cost algorithm to determine route
    - Distributed or centralized
  - All nodes
    - Like adjacent (except now use information from more nodes)

## Least Cost Algorithms

Dijkstra’s Algorithm

Bellman-Ford Algorithm

# Least Cost Algorithms

- Basis for routing decisions for almost all packet switching networks
  - If want to minimise number of hops, each link cost is 1
  - Or more typical: have link value inversely proportional to capacity
  - Other cost metrics can also be used
- Defines cost of path between two nodes as sum of costs of links traversed
  - In network of nodes connected by bi-directional links
  - Where each link has a cost in each direction (may be a different cost for each direction)
- For each pair of nodes, find the path with least cost
- Two common alternative algorithms for finding the paths:
  - Dijkstra algorithm
  - Bellman-Ford algorithm

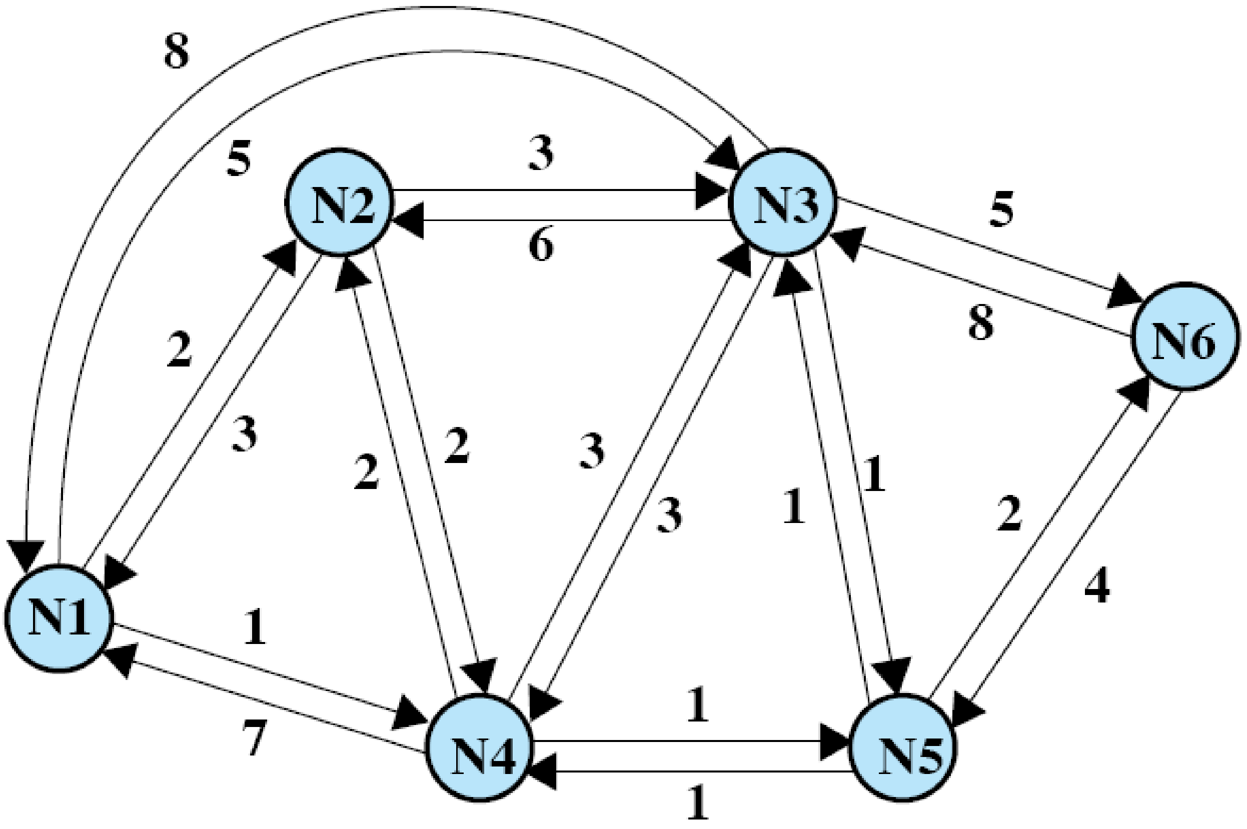
# Dijkstra's Algorithm

- Finds shortest paths from given source node *s* to all other nodes by developing paths in order of increasing path length
  - Algorithm runs in stages
    - Each time adding a node with next shortest path
  - Algorithm terminates when all nodes processed by algorithm (in set *T*)
- Define:
  - *N* = set of nodes in the network
  - *s* = source node
  - *T* = set of nodes so far incorporated by the algorithm
  - *w*(*i*,*j*) = cost from node *i* to node *j*
  - *L*(*n*) = cost of least-cost path from node *s* to node *n*

# Dijkstra's Algorithm

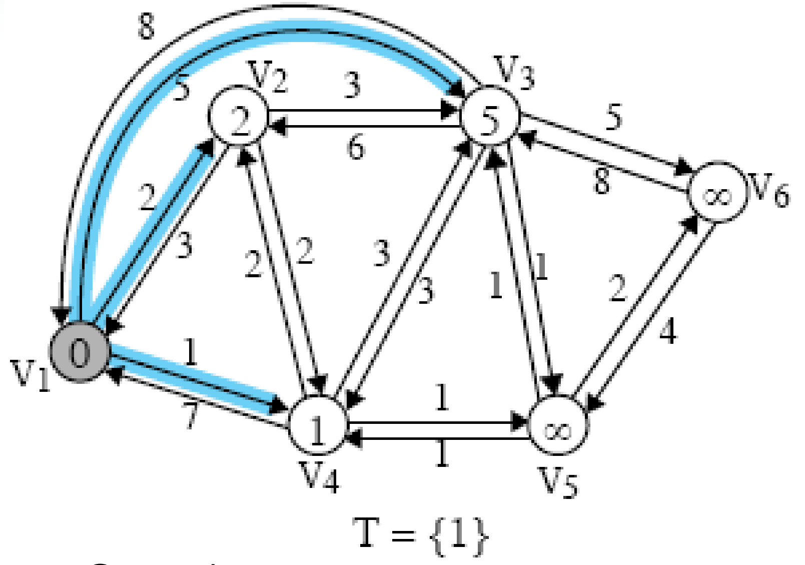
- Note: Steps 2 and 3 are repeated until $T = N$
  - Step 1 [Initialization]
    - $T = \{s\}$, set of nodes so far incorporated
    - $L(n) = w(s, n)$ for $n \neq s$; initial path costs to neighboring nodes are simply link costs
  - Step 2 [Get Next Node]
    - find neighboring node *x* not in *T* with least-cost path from *s*
    - incorporate node *x* into *T*
    - also incorporate the edge that is incident on that node and a node in T that contributes to the path
  - Step 3 [Update Least-Cost Paths]
    - $L(n) = \min[L(n), L(x) + w(x, n)]$ for all $n \notin T$
    - If latter term is minimum, path from *s* to *n* is path from *s* to *x* concatenated with edge from *x* to *n*

## Dijkstra's Algorithm Example

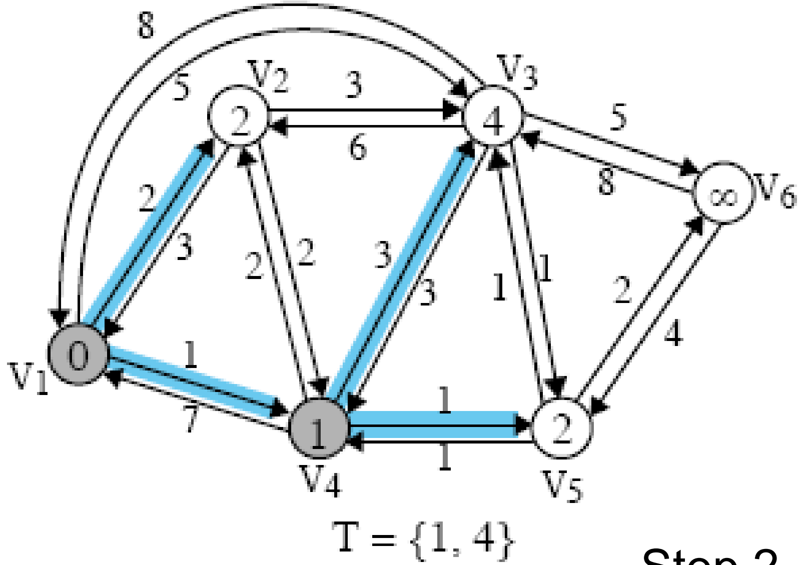


The example network. We will select node N1 as *s* first, and find the shortest paths to all other nodes. Then this process has to be repeated with the other nodes as the source, *s*.

# Dijkstra's Algorithm Example

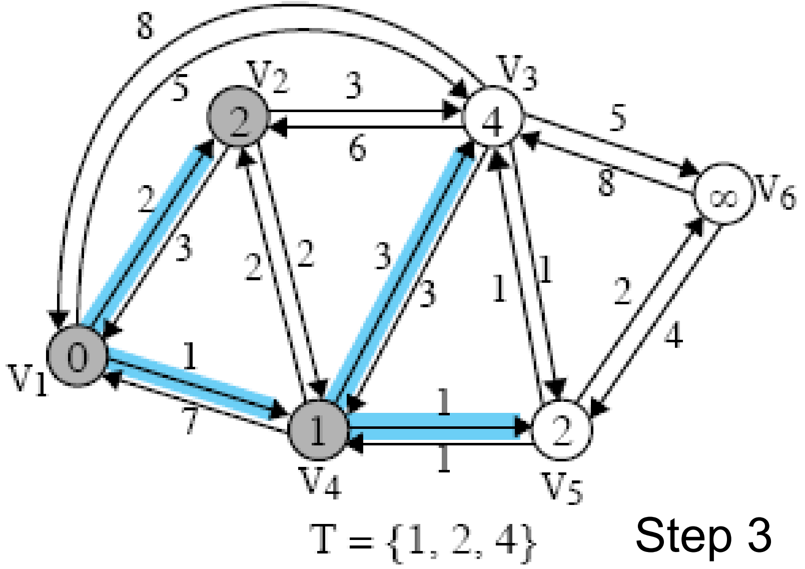


Step 1



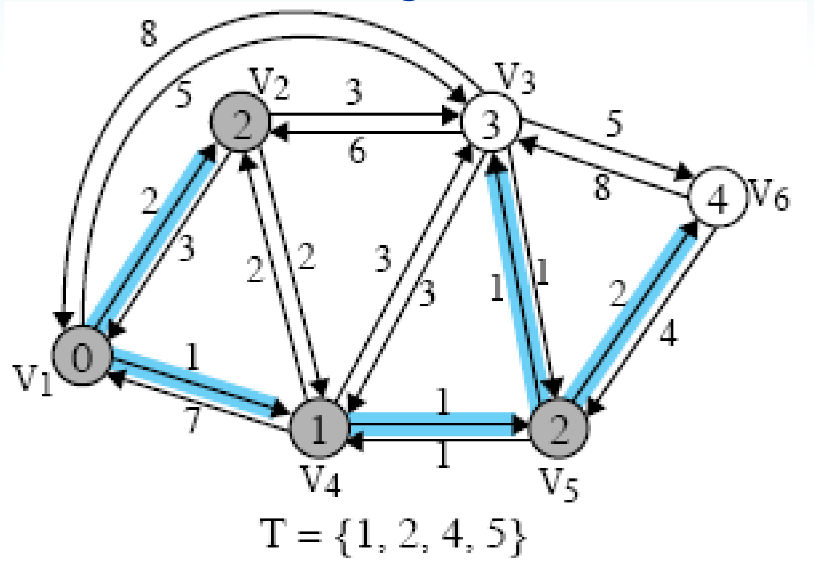
Step 2

The nodes are now labelled as $V_1$, $V_2$, …
The blue (thick) lines show the least-cost path from source $V_1$ to the other nodes under consideration. The numbers inside The circles give the current estimate of the least-cost from source $V_1$ to that node. For the nodes not yet covered, the least-cost is infinity (∞).

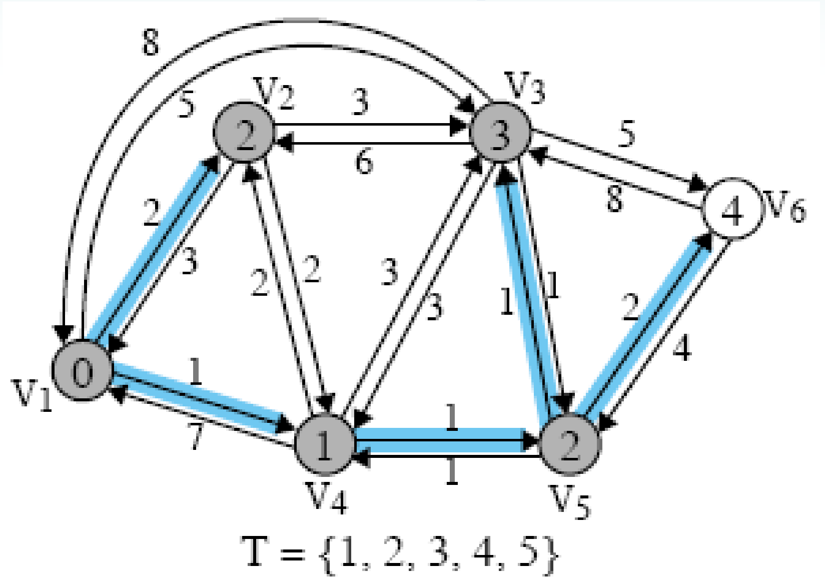


Step 3

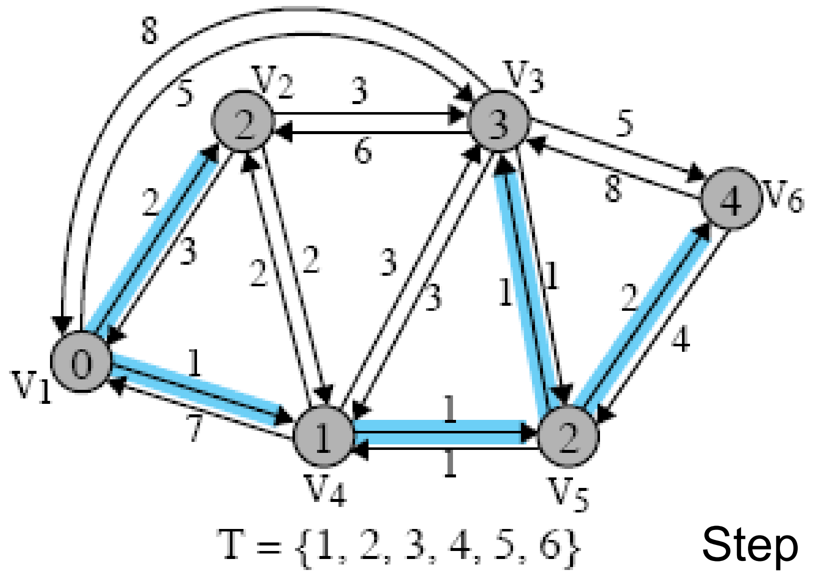
# Dijkstra's Algorithm Example



Step 4



Step 5



Step 6

# Dijkstra’s Algorithm Example

| Iter. | T | L(2) | Path | L(3) | Path | L(4) | Path | L(5) | Path | L(6) | Path |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | {1} | 2 | 1-2 | 5 | 1-3 | 1 | 1-4 | ∞ | - | ∞ | - |
| 2 | {1,4} | 2 | 1-2 | 4 | 1-4-3 | 1 | 1-4 | 2 | 1-4-5 | ∞ | - |
| 3 | {1, 2, 4} | 2 | 1-2 | 4 | 1-4-3 | 1 | 1-4 | 2 | 1-4-5 | ∞ | - |
| 4 | {1, 2, 4, 5} | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |
| 5 | {1, 2, 3, 4, 5} | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |
| 6 | {1, 2, 3, 4, 5, 6} | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |

This table is an alternative way to view the algorithm. In the first iteration, the source (N1) only knows of the cost to its direct neighbours (N2, N3 and N4). The least-cost, as well as the path with least-cost, are listed for each of the five other nodes (note the cost is ∞ for N5 and N6).

The algorithm then selects the node with least-cost (N4, which has cost of 1) and adds it to T. Then the second iteration is executed, where the least-cost path from N1 to each other node is updated. But know we also know the cost from N4 to its neighbours (including N5). So after the second iteration, we have a least-cost and path for N5 (the path is 1 to 4 to 5, which has a cost of 2).

Now we add N2 to T (or it could have been N5, since both have a least-cost of 2), and proceed with iteration 3.

The algorithm finishes when T contains all nodes: we have the least-cost paths from N1 to all other nodes.

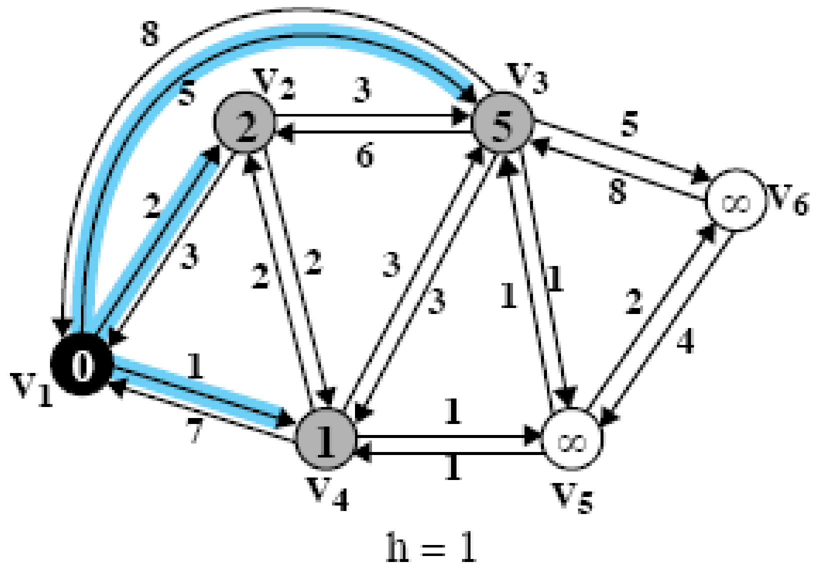
# Bellman-Ford Algorithm

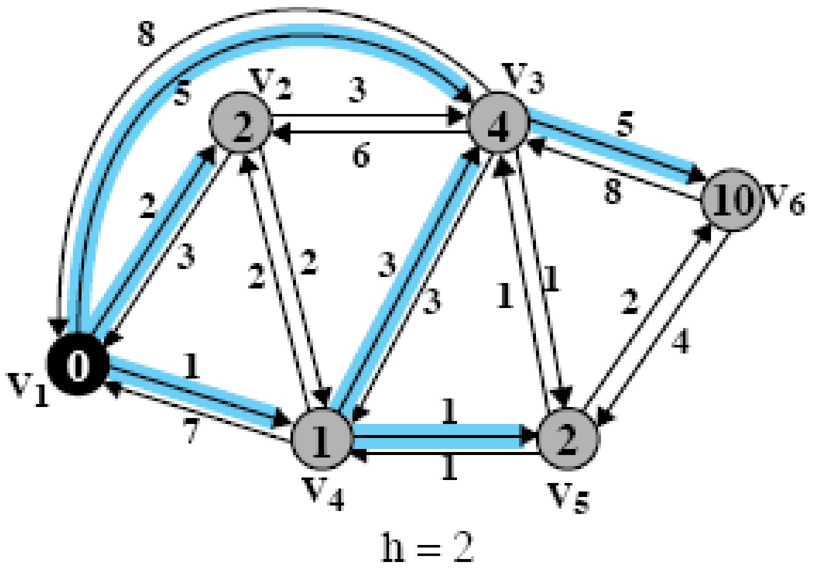
- Overview:
  - Find shortest paths from given node, subject to the constraint that paths contain at most one link
  - Then find the shortest paths with a constraint of paths of at most two links
  - And so on …
- Define:
  - $s$ = source node
  - $w(i, j)$ = link cost from node $i$ to node $j$
  - $w(i, i) = 0$
  - $w(i, j) = \infty$ if the two nodes are not directly connected
  - $w(i, j) \geq 0$ if the two nodes are directly connected
  - $h$ = maximum number of links in path at current stage of the algorithm
  - $L_h(n)$ = cost of least-cost path from $s$ to $n$ under constraint of no more than $h$ links

# Bellman-Ford Algorithm

- Step 1 [Initialization]
  - $L_0(n) = \infty$, for all $n \neq s$
  - $L_h(s) = 0$, for all $h$
- Step 2 [Update]
  - For each successive $h \geq 0$
    - For each $n \neq s$, compute: $L_{h+1}(n)=\min_j[L_h(j)+w(j,n)]$
  - Connect $n$ with predecessor node $j$ that gives minimum
  - Eliminate any connection of $n$ with different predecessor node formed during an earlier iteration
  - Path from $s$ to $n$ terminates with link from $j$ to $n$

# Bellman-Ford Algorithm Example





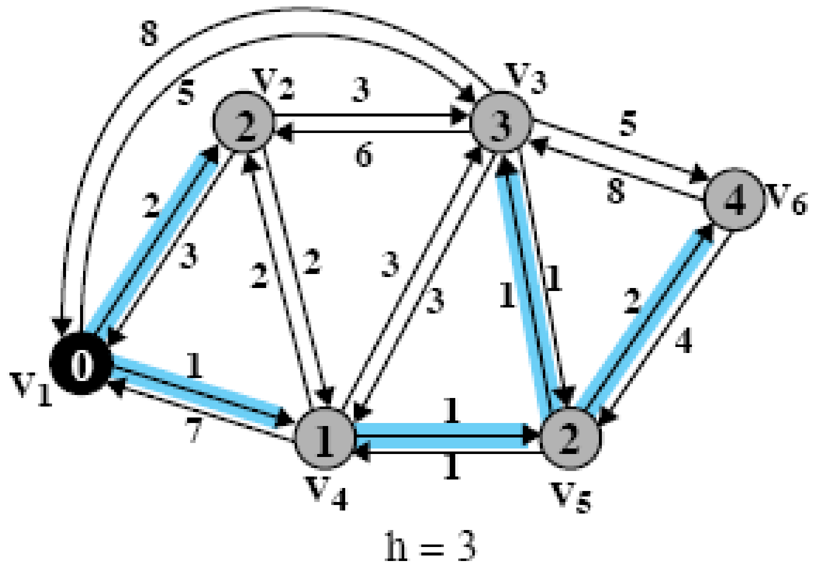
Step 1

Step 2

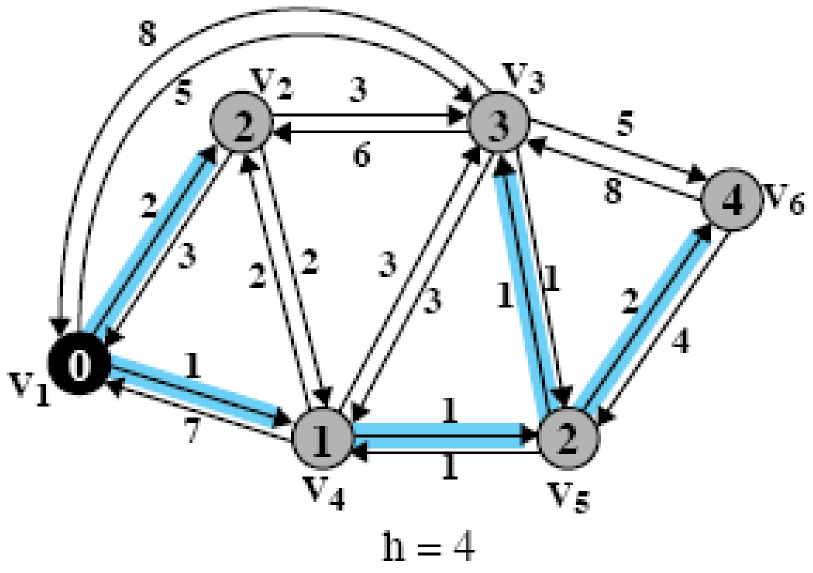
For all the paths that are 1 hop, find the shortest path to other nodes. In this example, only nodes V2, V3 and V4 can be reached in 1 hop.

For all the paths that are 2 hops, find the shortest path to other nodes.

# Bellman-Ford Algorithm Example



Step 3



Step 4

## Bellman-Ford Algorithm Example

| h | $L_h(2)$ | Path | $L_h(3)$ | Path | $L_h(4)$ | Path | $L_h(5)$ | Path | $L_h(6)$ | Path |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $\infty$ | - | $\infty$ | - | $\infty$ | - | $\infty$ | - | $\infty$ | - |
| 1 | 2 | 1-2 | 5 | 1-3 | 1 | 1-4 | $\infty$ | - | $\infty$ | - |
| 2 | 2 | 1-2 | 4 | 1-4-3 | 1 | 1-4 | 2 | 1-4-5 | 10 | 1-3-6 |
| 3 | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |
| 4 | 2 | 1-2 | 3 | 1-4-5-3 | 1 | 1-4 | 2 | 1-4-5 | 4 | 1-4-5-6 |

# Comparison

- Results from two algorithms agree
  - They both find the same least-cost paths
- Bellman-Ford Algorithm
  - Calculation for node *n* needs link cost to neighbouring nodes plus total cost to each neighbour from *s*
  - Each node can maintain set of costs and paths for every other node
  - Can exchange information with direct neighbours
  - Can update costs and paths based on information from neighbours and knowledge of link costs
- Dijkstra's Algorithm
  - Each node needs complete topology
  - Must know link costs of all links in network
  - Must exchange information with all other nodes
- Evaluation depends on processing time of algorithms, amount of information they exchange, and implementation details
  - We will see later how the algorithms are used in the Internet