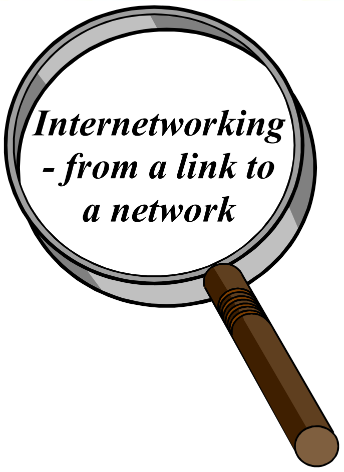
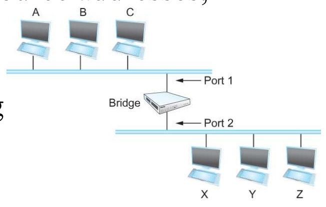


Internetworking - from a link to a network

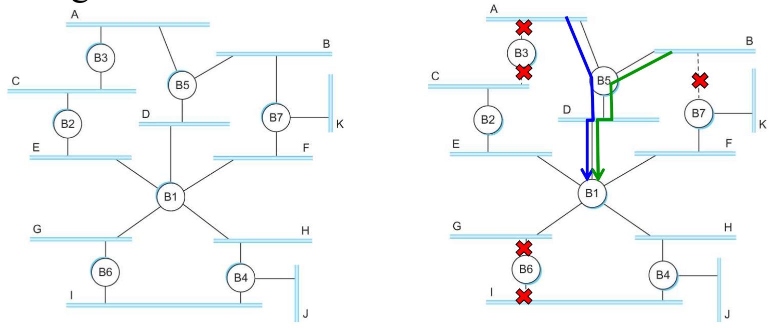
Repeaters, Bridges, Routers, Gateways

- *Repeaters:* operates on the physical layer by retransmitting bits from one port to the other ports
- *Bridge:* operates on the frames, by forwarding frames from one port to the other ports; in a LAN, implements the MAC-layer protocol
- *Learning Bridge:* filters frames based on MAC address;
  - the learning is done by observing the source addresses;
  - initially the forwarding table is empty
  - the entries include a timeout value
  - considers as an optimization of routing



Repeaters, Bridges, Routers, Gateways

❖ *Spanning Tree Bridges:* To correct the problem of loops in bridged network



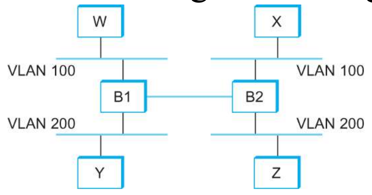
The selection is made based on the length of the path, with ties broken by the bridge ID. What problems this may create?

Repeaters, Bridges, Routers, Gateways

❖ *Broadcast* and *Multicast Bridges*
   ❖ Note: practically, multicast is done as broadcast

❖ Extended LANs may practically include few tens of LANs, but this is not a *scalable* solution. (This is where routers come into the picture.)

❖ *Virtual LANs (VLANs):* an Extended LAN is partitioned into *VLANs* by assigning colors and limiting forwarding within any *VLAN.*
   ❖ *VLANs* are easy to reconfigure



Datagrams

- Datagram networks (packet-switched networks):
- *Connectionless* (no state needed before data is transmitted).
- No guarantee of delivery (the network can drop packets or to the destination may be down).
- Each packet is forwarded independently through the network.
- Router or link failure needs to be mitigated.

Virtual Circuit Switching

- Virtual Circuit (VC) is a *connection-oriented* communication model.
- I.e., it is necessary to establish a *connection state* along the source-destination path before communication commences (*c.f.* circuit-switching).
- *VC table* stores connection entries; one entry per connection
- In general, a VC entry consists of:
  - Incoming *VC identified (VCI)*
  - Outgoing *VC identified (VCI)*
  - Incoming interface (link)
  - Outgoing interface (link)

An *VCI* is unique on each interface and has local significance only

Virtual Circuit Switching

- Virtual Circuit (VC) can be *Permanent (PVC)* or *Switched (Signalled) VC (SVC).*
- All the advantages/disadvantages of *Circuit Switching* (vs. *Packet Switching)* apply to *VC* vs. Datagram networks.
- X.25 was (an early) VC-based network standard:
  - Buffers are allocated at each node for a VC
  - Connection set-up is rejected at a node when there is insufficient buffer space available
  - Sliding-window protocol (+flow control) for each VC
- Buffer allocation allows to *hop-by-hop flow control.*
- VC-based networks allow VC-based QoS differentiation.

Virtual Circuit Switching

- Examples of Virtual Circuit networks:
  - X.25
  - Frame Relay
  - *Asynchronous Transfer Mode* (*ATM*)
  - *Virtual Private Networks (VPNs)*

Marconi ForeRunnerLE 25 ATM PCI network interface card

- ATM as an example of VC-based networks – 53-byte *cells*.

4	8	16	3	1	8	384 (48 bytes)
GFC	VPI	VCI	Type	CLP	HEC (CRC-8)	Payload

- The *cell* size as a compromise (*c.f.* our discussion of "optimal" packet size)

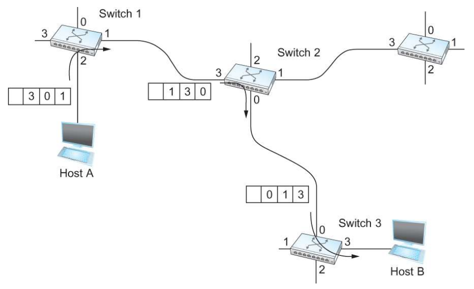
Virtual Circuit Switching

4	8	16	3	1	8	384 (48 bytes)
GFC	VPI	VCI	Type	CLP	HEC (CRC-8)	Payload

- ❖ GFC = Generic Flow Control (4 bits) (default: 4-zero bits)
- ❖ VPI = Virtual Path Identifier
- ❖ VCI = Virtual Channel identifier (16 bits)
- ❖ PT = Payload Type (3 bits)
  - ❖ PT bit 3 (msbit): Network management cell. If 0, user data cell; if 1 it's OA&M (Operation, Administration, and Management) cell.
  - ❖ PT bit 2: Explicit Forward Congestion Indication (EFCI); 1 = network congestion experienced (set to 0 by the source and to 1 by congested switch on the cell path)
  - ❖ PT bit 1 (lsbit): ATM user-to-user (AAU) bit. Used by AAL5 to indicate packet boundaries.
- ❖ CLP = Cell Loss Priority (1-bit): 0=high priority
- ❖ HEC = Header Error Control (8-bit CRC, polynomial = $X^8 + X^2 + X + 1$): correct single error and detect multiple errors

Source Routing

- ❖ Although interface number could be used as identified, the node ID is a better choice.



- ❖ Main advantage of source routing: ease of processing
- ❖ Main disadvantage of source routing: the header size depends on the path length
- ❖ Other disadvantage: static routing, etc

Source Routing

- *Source Routing* can be used in datagram (hybrid) and *VC* networks (for set-up request routing).
- *Strict Source Routing* : every node on the path is included in the route
- *Loose Source Routing*: only a set of nodes to be traversed is included in the route

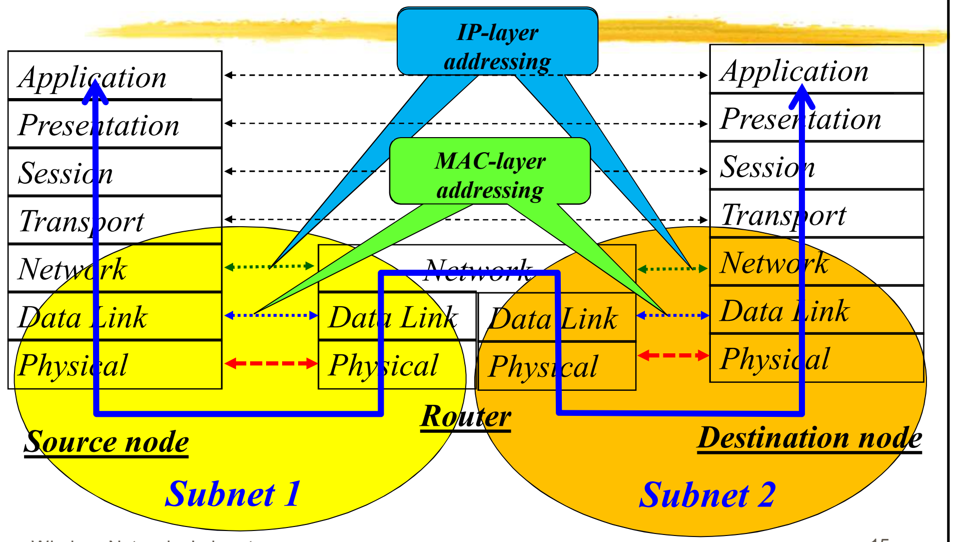
Forwarding at Routers

- How the source node gets its datagram to the destination node?
  - If the source node is on same subnet (LAN) as the destination node, the source node sends the datagram directly to the destination node using MAC-layer addressing (how it obtains the MAC layer address of the destination node? ARP – *Address Resolution Protocol* – RFC 826)
  - Otherwise, the source node sends the datagram to a router on the same subnet as the source node.
  - The router will then forward the datagram to a router on the next subnet …. and so on until the datagram arrives at router on same subnet as the destination node.
  - Then, the "last router" sends the datagram directly to destination node using MAC layer addressing.

Forwarding at Routers

- For this process to work, at least the following requirements need to be satisfied:
  - A node (including the designated router) needs to know the MAC address of other nodes on their subnet (i.e., need to know how to "translate" an IP address of such nodes to the corresponding MAC address).
  - Every node needs to know the IP address of the router on its subnet
  - Every router needs a "routing table" to tell it which neighboring router (i.e., network) to forward a given datagram to.

Routing Across Networks



Forwarding vs. Routing

- What we have discussed until now is the concept of *forwarding* …. i.e., how a router forwards datagrams.
- The concept of *routing* necessitates solving a global problem in a network … finding a *route.*
- I.e., forwarding is a local process, while routing is a global process.
- Finding routes in a network is essentially a graph-theory problem.
- There are two basic approaches:
  - *Distance Vector* protocols
  - *Link State* protocols

Traditional Routing Protocols - the Distance-Vector Algorithms

- The Distance-Vector algorithm is also called *distributed Bellman-Ford* and *Ford-Fulkerson* routing algorithms.
- Each node knows the “distance” to every neighbor *i*, *c(i).*
- Each node maintains a table with an entry for every destination node in the network, which specifies the metric (e.g., time or distance) and the direction-vector (i.e., neighbor node) to reach the destination node.
- The tables are updated by periodic exchanges of information with the neighbors.
- I.e., periodically, each node exchanges its table with all its *N* neighbors and receives the neighbors’ tables.

The Distance-Vector Algorithms

- Each node then updates its distance-vector entry to every destination *j*, *D(j)*. If *D(i,j)* is the distance to destination *j,* as reported by neighbor *i,* then the updated distance to *j* is:

$$D(j) = \min_{i=1}^{N} [D(i,j) + c(i)]$$

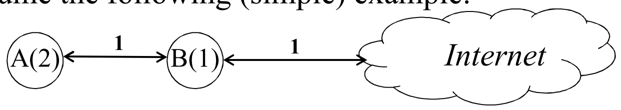
- Convergence … does the algorithm converge in light of topological changes?
- The new vector is the value of *i* that minimizes the above equation.
- In the Distance Vector algorithm, “*good news propagates fast, while bad news are delayed.”*
- This is also referred to as the *Count to Infinity Problem.*
- It takes *n* cycles to propagate the information about a new link across a path of length *n.*

The Distance-Vector Algorithms (con't)

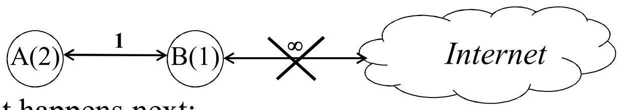
- Upon a node failure, the distance to the failed node at its neighbor increases by 1 in each cycle, eventually reaching the maximum value (theoretically, infinity).
- Examples of Distance Vector algorithms are *RIP* (*Routing Information Protocol)* and *HELLO*.
- The disadvantages of Distance Vector:
  - poor scalability
    - message size is proportional to the number of destinations
    - every node sends the periodic exchange messages
  - the Count-to-Infinity Problem
- Link State Routing (*Shortest Path First - SPF*) is an alternative to Distance Vector routing

The Count to Infinity Problem

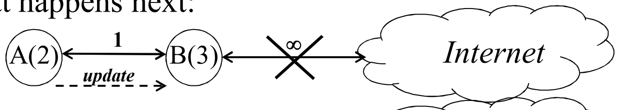
- Assume the following (simple) example:

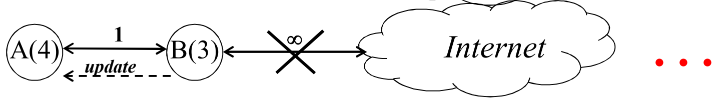


- Now assume that the link B-to-Internet fails:

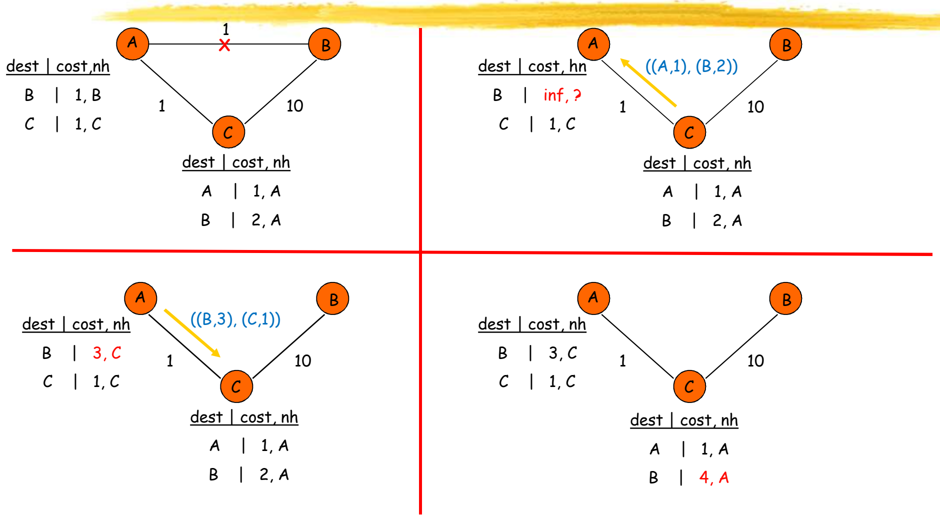


- What happens next:





The count to infinity problem



Copyright ©by Zygmunt J. Haas, 2017

The Count to Infinity Problem (con't)

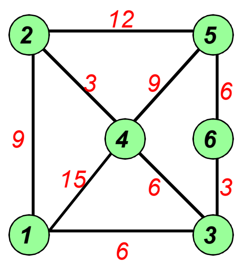
- What is the problem?
- Possible solutions to the *Count to Infinity* problem:
  - *Split Horizon:* nodes (routers) never advertise the cost of a destination back to its next hop (this is where it has learnt it from).
  - *Poison Reverse:* advertise such routes to have infinity cost.
- Those improvements do not solve the problem in more complicated topologies … think of some examples when these improvements fail.
- A possible improvement (not a solution!) is to limit the counting by declaring some number to be infinity (e.g., 16).

Copyright ©by Zygmunt J. Haas, 2017

The Bellman-Ford Algorithm: An example

⌘ Consider the following example network:

From/To	1	2	3	4	5	6
1	0	9	6	15	∞	∞
2	9	0	∞	3	12	∞
3	6	∞	0	6	∞	3
4	15	3	∞	0	9	∞
5	∞	12	∞	9	0	6
6	∞	∞	3	∞	6	0



⌘ We will label each node by *{(k, D(j))}*, where *D(j)* is the current minimal cost from the labeled node to the destination *j*, and *k* is the neighbor node along which this minimal cost can be achieved; value of "*U*" means "undefined".

The Bellman-Ford Algorithm: An example (con't)

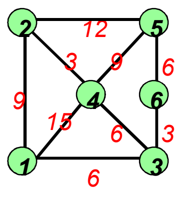
⌘ The following are the steps in the execution of the algorithm:

Iteration # 1:

To node

From \To	1	2	3	4	5	6
1	(0,0)	(2,9)	(3,6)	(4,15)	(U,∞)	(U,∞)
2	(1,9)	(0,0)	(U,∞)	(4,3)	(2,12)	(U,∞)
3	(1,6)	(U,∞)	(0,0)	(4,6)	(U,∞)	(6,3)
4	(1,15)	(2,3)	(3,6)	(0,0)	(5,9)	(U,∞)
5	(U,∞)	(2,12)	(U,∞)	(4,9)	(0,0)	(6,6)
6	(U,∞)	(U,∞)	(3,3)	(U,∞)	(5,6)	(0,0)

From node



The Bellman-Ford Algorithm: An example (con't)

⌘ The following are the steps in the execution of the algorithm:

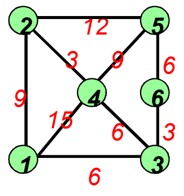
To node

Iteration # 4:

From \To	1	2	3	4	5	6
1	(0,0)	(2,9)	(3,6)	(2,12)	(3,15)	(3,9)
2	(1,9)	(0,0)	(4,9)	(4,3)	(2,12)	(4,12)
3	(1,6)	(4,9)	(0,0)	(4,6)	(6,9)	(6,3)
4	(2,12)	(2,3)	(3,6)	(0,0)	(5,9)	(3,9)
5	(6,15)	(2,12)	(6,9)	(4,9)	(0,0)	(6,6)
6	(3,9)	(3,12)	(3,3)	(3,9)	(5,6)	(0,0)

From node

Iteration # 4 = Iteration # 3 ⇒ STOP.



Copyright ©by Zygmunt J. Haas, 2017

The count to infinity problem

⌘ Partial solutions
- ❖ Use a small value to represent infinity
- ❖ Split horizon
  - ➢ C does not advertise route to A
    - • Why: C uses A to reach B
- ❖ Poison reverse
  - ➢ C advertises route to A with infinite distance
- ❖ Work for two node loops
  - ➢ Does not work for loops with more nodes (Convince yourself using an example)

⌘ How to avoid loops (i.e., perfect solution) ?
- ❖ Each advertisement carries the entire path
- ❖ BGP uses this approach

Copyright ©by Zygmunt J. Haas, 2017

The Routing Information Protocol (RIP)

- RIP is an example of a *distance vector* protocol.
- RIP uses *hop count* as cost metric.
  - infinity value is declared as 16 hops; this limits the network size.
  - it includes the split horizon with poison reverse improvements.
- Routers send update vectors every 30 seconds
  - link failures trigger updates
  - uses timeout of 180 seconds to detect failures
- RIPv1 is specified in RFC 1058 and RIPv2 (which adds authentication) in RFC 1388. See www.ietf.org/rfc web site.
- Note that RIP is an Interior Gateway Protocol, suitable for small- to medium-size networks. Why? (hop-count as a metric; 16-hop limit, etc).

Traditional Routing Protocols - the Link State Algorithms

- Link State Routing requires all the network nodes to know the complete network topology
- Each node learns the liveness of the links to its neighbors (i.e., who their neighbors are). This can be done, for example, by exchanging “HELLO” beacons and using the *k-out-of-n* rule.
- Periodically, each node sends the status of the links to its neighbors (i.e., the list of its neighbors) to all the other network nodes.
- Upon receipt of an update at a node, the topology of the network, as viewed by the node, is updated.

The Link State Algorithms (con't)

- Shortest path algorithm (e.g., the *Dijkstra algorithm)* is then used to compute a path to any network destination from the node in question.
- Convergence is guaranteed, as each node acts independently on the topological information.
- The size of the message does not depend on the number of network nodes (i.e., SPF is more *scalable*).

The Open SPF Protocol - OSPF

- Goals of OSPF:
  - open standard to replace proprietary protocols
  - type-of-service routing
  - load balancing
  - allows network partition to self-contained subsets termed *areas*
  - variety of authentication schemes
  - host- and network-specific routes
  - Uses the notion of the designated router to minimize the amount of periodic broadcast link status
  - allows virtual network topology
  - allows the exchange of information from external sites.

The Dijkstra's Algorithm

- Goal: Find the shortest route from a given node (source) to all other network nodes.
- The algorithm works by finding paths of increasing order; in the *k-th* iteration, the paths to the *k* closest nodes to the source have been identified. These nodes are represented as the set *M.*
- Nomenclature of the Dijkstra's Algorithm:
  - *N:* set of node in the network
  - *s:* source node
  - *M:* set of nodes *k* closest nodes identified in the *k-th* iteration of the algorithm

The Dijkstra's Algorithm (con't)

- *d(i,j):* cost of link from node *i* to node *j;*
  - if node *i* and *j* and directly connected ⇒ $d(i,j) \geq 0$
  - if nodes *i* and *j* are not directly connected ⇒ $d(i,j)=\infty$
  - where $d(i,i)=0$
- D_n: the smallest cost from the source to node *n* that has been discovered until now by the algorithm

The Dijkstra's Algorithm (con't)

⌘ **Formal Description of the algorithm:**

(1) Initialize:

- $M=\{s\}$
- for each $n \neq s \Rightarrow D_n=d(s,n)$

(2) Find $w \notin M$, such that $D_w=min\ D_j$ for $j \notin M$

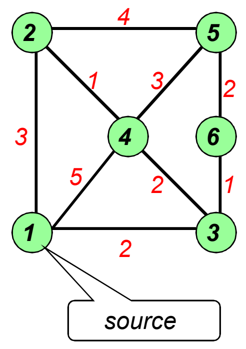
(3) $D_n=min[D_n,D_w+d(w,n)]$ for all $n \notin M$

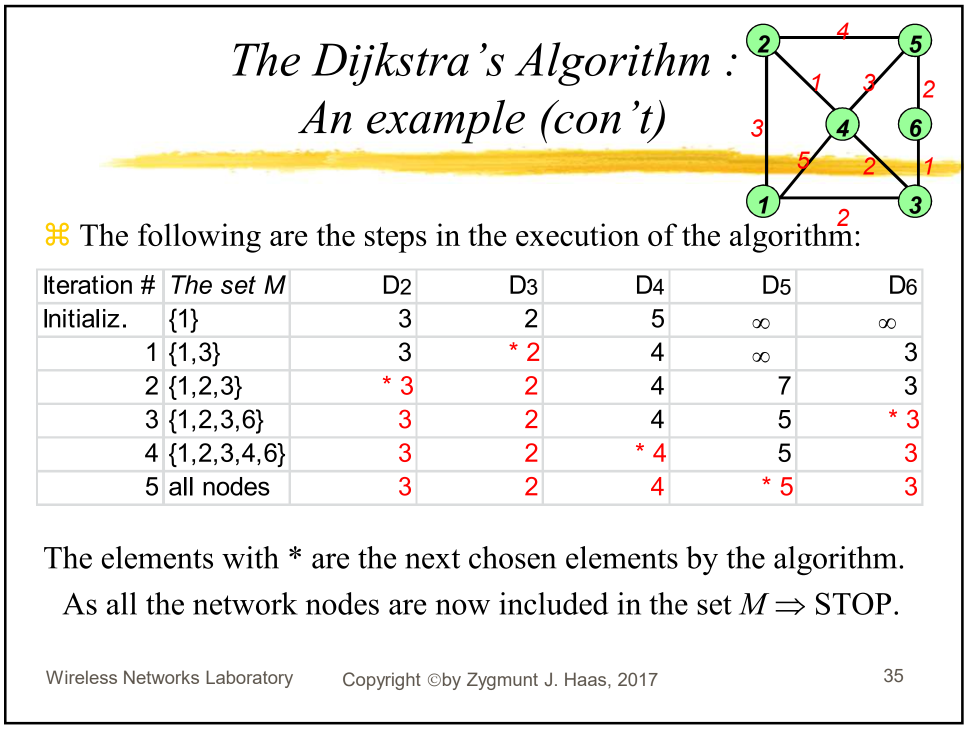
(4) Repeat (2) and (3) until M contains all the network nodes.

The Dijkstra's Algorithm : An example

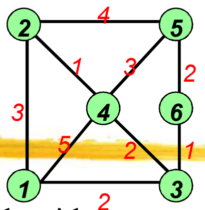
⌘ Consider again the following example network:

From/To	1	2	3	4	5	6
1	0	3	2	5	∞	∞
2	3	0	∞	1	4	∞
3	2	∞	0	2	∞	1
4	0	3	∞	0	9	∞
5	∞	4	∞	3	0	2
6	∞	∞	1	∞	2	0





The Dijkstra's Algorithm : An example (con't)



⌘ The following are the steps in the execution of the algorithm:

Iteration #	*The set M*	D2	D3	D4	D5	D6
Initializ.	{1}	3	2	5	∞	∞
1	{1,3}	3	* 2	4	∞	3
2	{1,2,3}	* 3	2	4	7	3
3	{1,2,3,6}	3	2	4	5	* 3
4	{1,2,3,4,6}	3	2	* 4	5	3
5	all nodes	3	2	4	* 5	3

The elements with * are the next chosen elements by the algorithm.
As all the network nodes are now included in the set *M* ⇒ STOP.