Routing

Ghazi Bouabene
University of Basel
Cs 321 – HS 2011

Overview

Intro: The network environment

Part 1: Association

Part 2: Topology Management

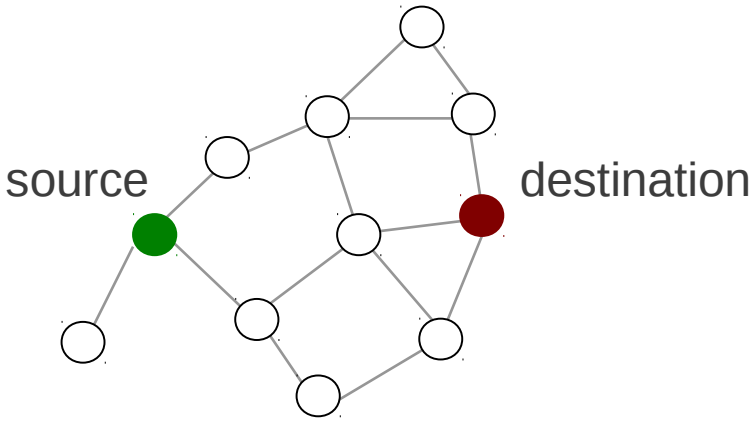
Part 3: Routing

General about network routing

- Definitions
- Classifications of routing approaches
- Route Discovery protocols
- Examples of routing protocols
- Greedy routing

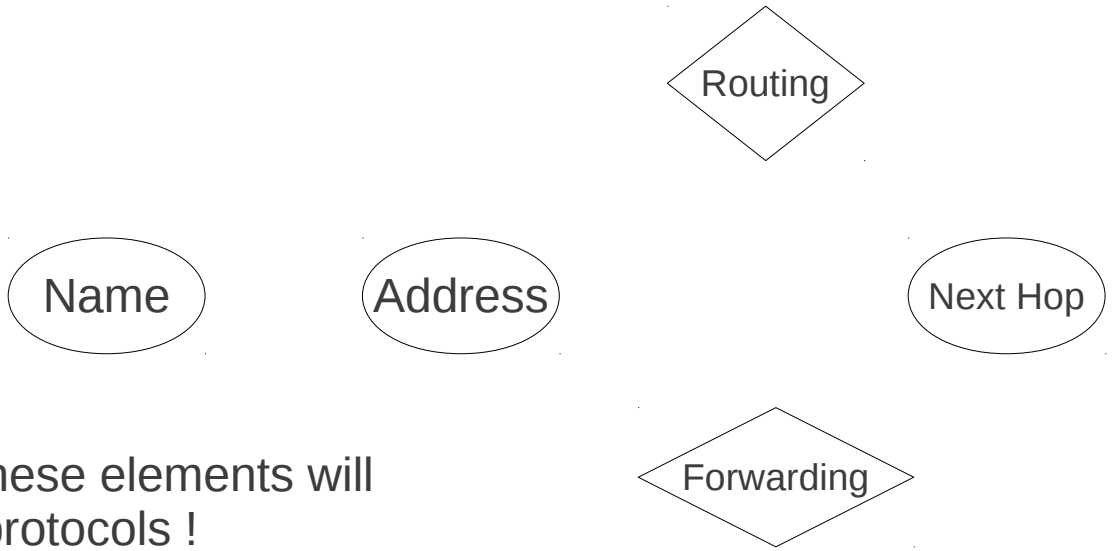
Part 4: Communication

Overview



Starting from a Name or Address, which Next-hop (neighbor), should the source send the message to ?

Main Components to decision :

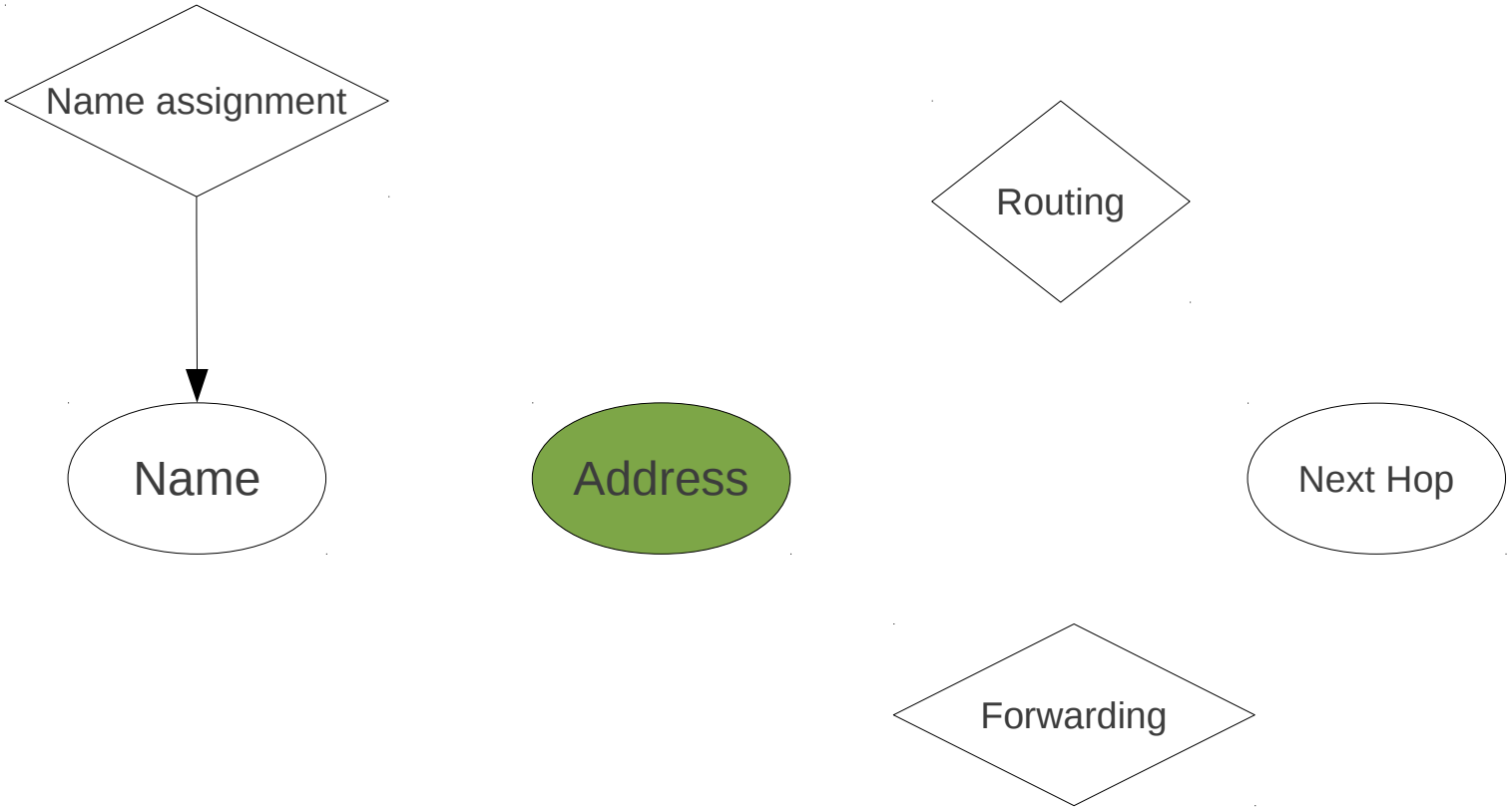


Different paths between these elements will result in different routing protocols !

What is a “Name” ?

- The name of a resource indicates what we seek
- Names can be structured (or not) : e.g. flat or hierarchical
- They can carry semantics : inform who is the owner, indicate the type of content/service ..
- Names can be assigned :
  - Statically (humanly) : by organizations (ICANN), governments, parents ..
  - Autonomously : from simple counting to using feature extraction schemes

Overview



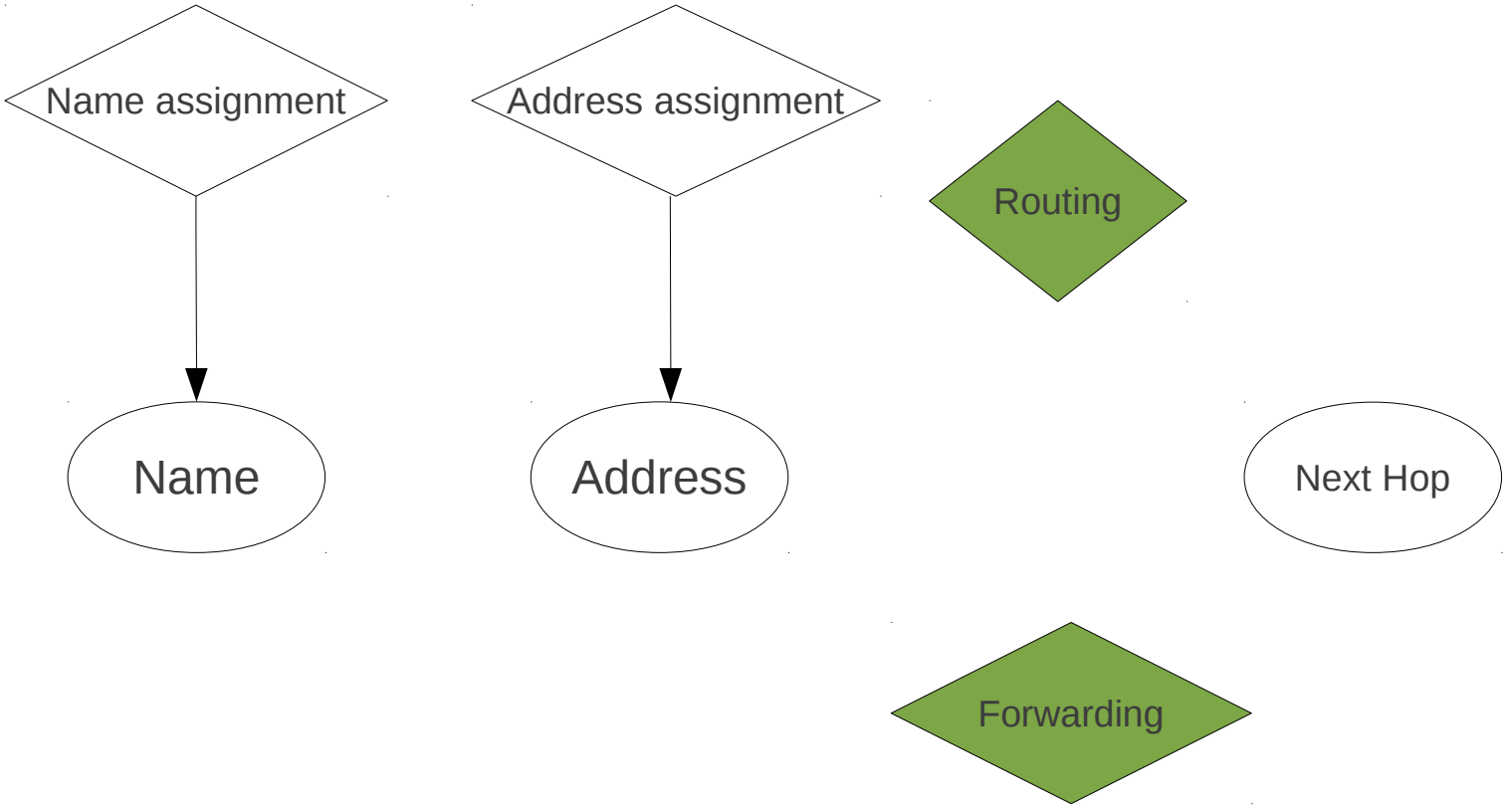
What is an “Address” ?

- The address of a resource indicates where it is
- Address should be assigned so as to reflect locality in the network:

  2 nodes that are close to each other in the network, should be attributed addresses that are *similar* to each other.

  - Statically (humanly) : by organizations e.g. IANNA for IP addresses
    - Problem : it is hard to cope with changes in the network. With time addresses loose their location information
  - Dynamically :
    - Much more flexible towards changes in the network topology
    - Hosts and applications have to cope with continuous changes in their network addresses

Overview


Name assignment
Address assignment
Routing
Name
Address
Next Hop
Forwarding

Routing and Forwarding

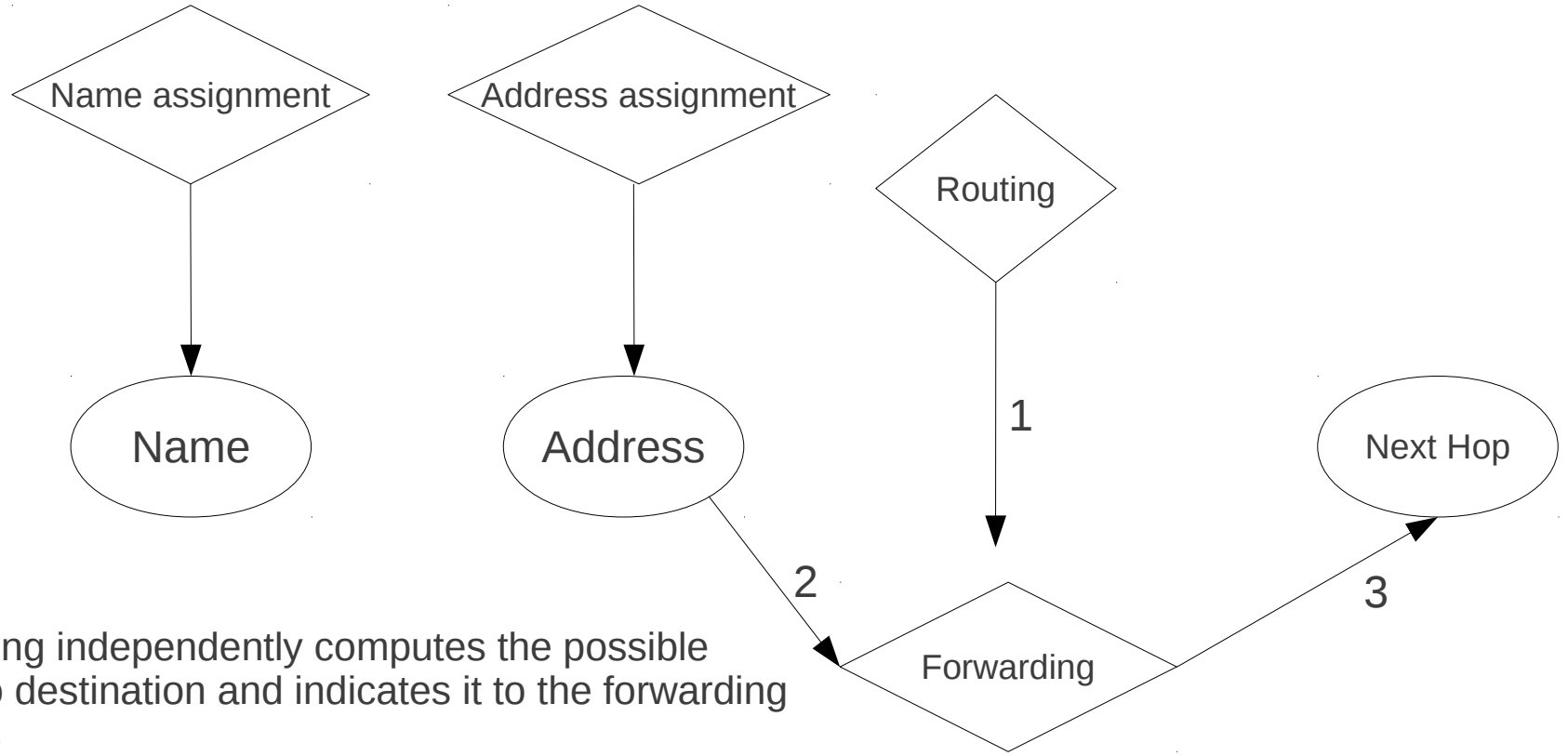
- A route tells us how to reach a network resource
- Routing is the process of discovering the network and finding means of reaching resources
- Often confused with forwarding :
  - Routing is the step previous to forwarding.
  - Routing prepares the information used for forwarding.
- Forwarding is the simple act of executing routing instructions
  - Example : To turn left when your GPS guiding system tells to turn left :)

Routing and Forwarding

- The frequency of interaction between forwarding and routing depends on whether we use **proactive** or **reactive** routing.

- Proactive routing means that the information required for forwarding is prepared before any communication needs
  - Routing paths are (re-)computed every time the topology or node associations change
  - Maintain information to all possible destinations (overhead increase with number of nodes)
  - Requires periodic exchange of routing information among peer nodes
  - Better for frequent communications with multiple nodes

Pro-active routing



1) Routing independently computes the possible paths to destination and indicates it to the forwarding process

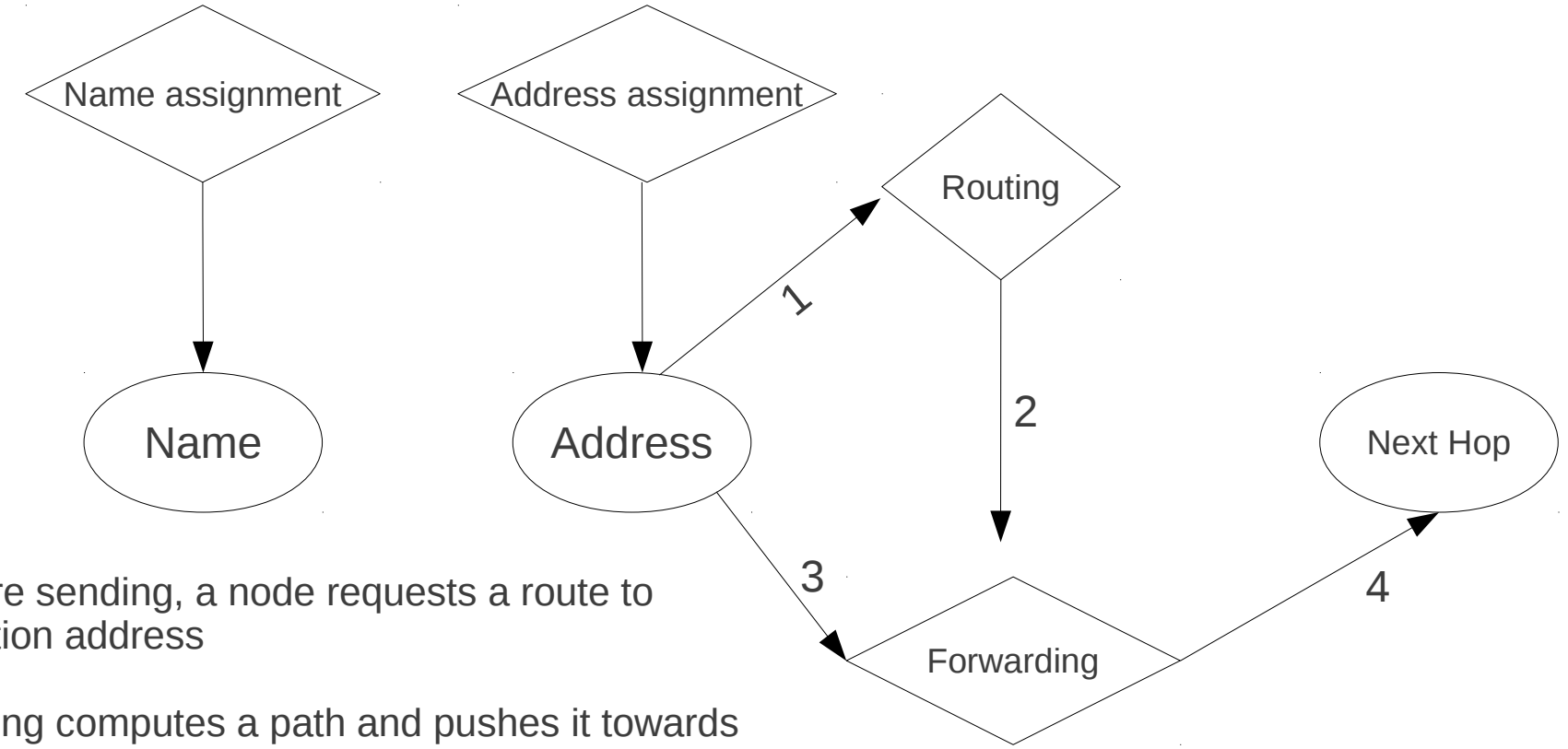
2) When sending towards an address a host can directly proceed to the forwarding step

3) Routing instruction are followed and packet is forwarded to the next hop

Routing and Forwarding

- The other case is that of **reactive routing** (aka on-demand routing)
  - No routing information is computed before the communication
  - And hence there is also no prior forwarding information.
  - Routing paths are computed when a communication is requested
  - Maintain information only to destinations of active communications (overhead increase with number of communications)
  - Better for topologies that change very frequently

Reactive routing



1) Before sending, a node requests a route to destination address

2) Routing computes a path and pushes it towards forwarding

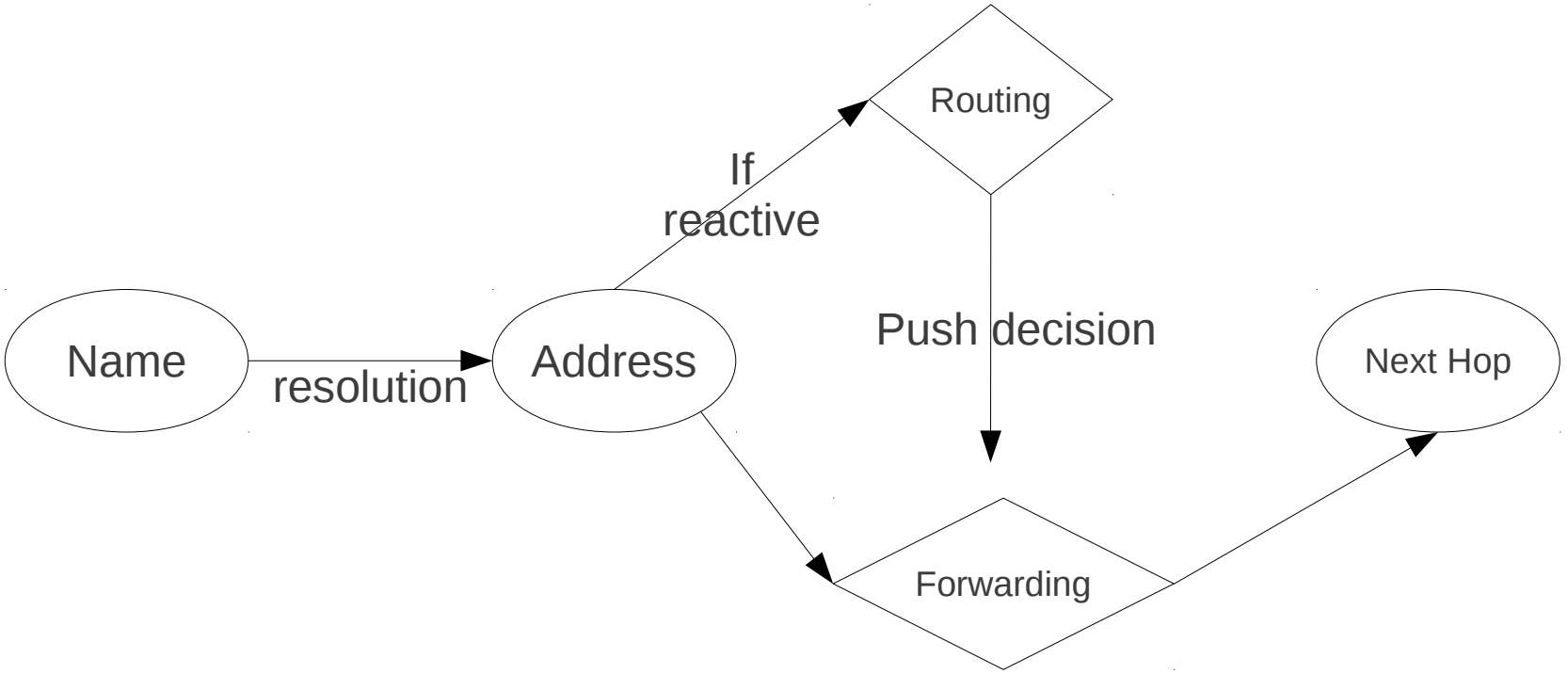
3) nodes can then proceed and hand its packets to the forwarding phase

4) Routing instruction are followed and packet is forwarded to the next hop

Name-based vs address-based routing

- In current Internet, routing is address based :
  - Given a name of a resource, it is first transformed into an address by what is called a “resolution” process
- Resolution can be done in many different ways : querying a distributed database (DNS), asking all present nodes (flooding)
- The address is then the element used in both the routing and the forwarding procedure

address-based routing

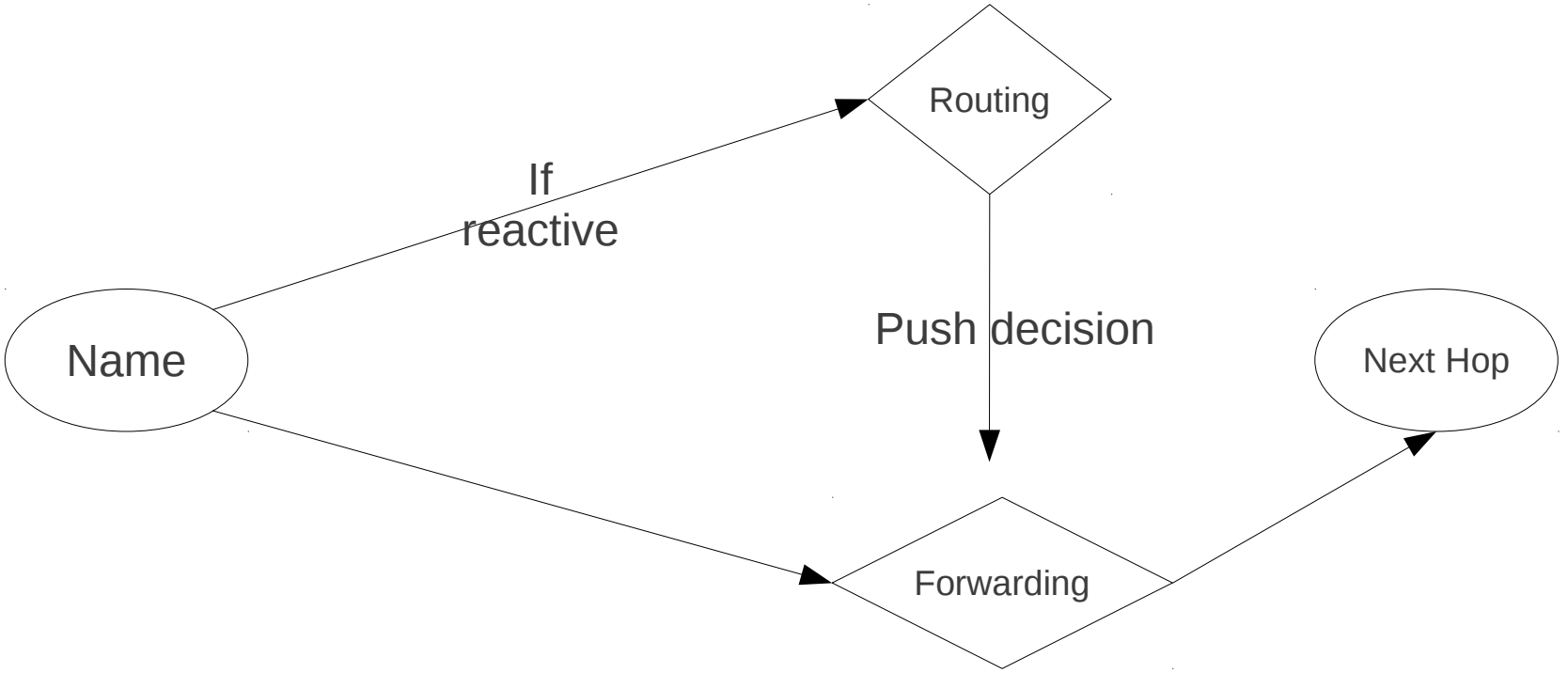


Note : a communication can also be started using the address directly without necessarily requiring a name

Name-based vs address-based routing

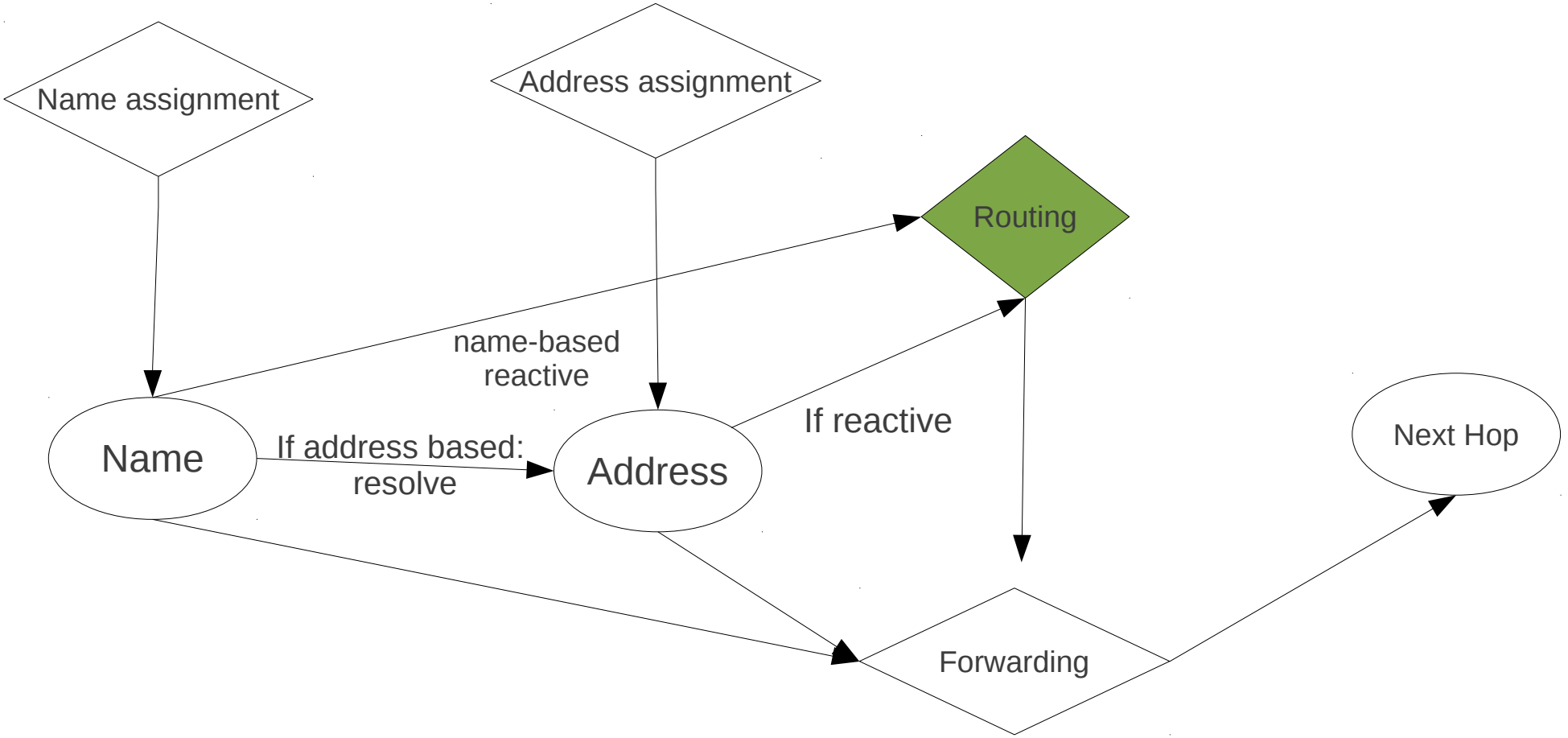
- Another possibility is to directly use the resource names for both routing and forwarding
- Advantages:
  - Allows for more flexible types of routing:
  - The possibility of retrieving a resource from many points and not necessarily a specific one
  - Much simpler and intuitive support for mobility : send a message to a person no matter her current location !
- Disadvantage/challenge :
  - The absence of addresses that localize the resource complicates the routing procedure

name-based routing



Note : In such a system, routing will do the job of locating the resource and hence addresses might not be needed at all

Overview


Name assignment
Address assignment
Routing
name-based
reactive
Name
If address based:
resolve
Address
If reactive
Next Hop
Forwarding

Steps of Routing

- Routing proceeds in 2 steps :
  - 1st step is the path discovery
  - 2nd step is pushing the routing information towards forwarding
- For the discovery step, there are generally 3 different families of protocols :
  - Path-vector protocols
  - Link-state protocols
  - Distance-vector protocols
- Note that the routing information can also be provided statically (human input), but that's boring

Path-vector protocols (1/4)

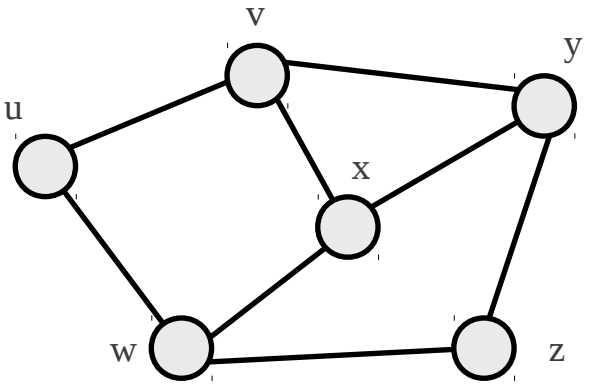
- Main idea :
  - Discovery packets traveling around the network, keep a record of the traversed nodes (or edges)
  - Paths to a resource of interest can therefore be simply obtained by looking at the path registered in the packet
- Example :

<u>How **u** learns a path towards **z** :</u>

1) **z** sends a HELLO packet to all his neighbors

2) Each node receiving a HELLO packet pushes its node identifier (or its edge identifiers) into the packet's path trace

3) When **u** receives a HELLO packet (originated from **z** or relayed by **z**) it can retrieve a path to **z** from the packet trace



Path-vector protocols (2/4)

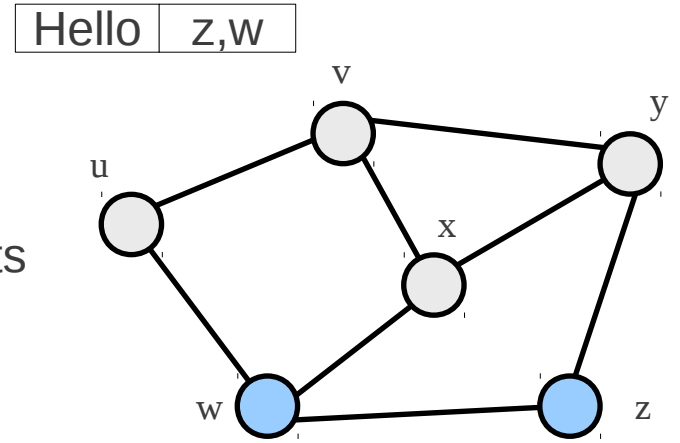
- Main idea :
  - Discovery packets traveling around the network, keep a record of the traversed nodes (or edges)
  - Paths to a resource of interest can therefore be simply obtained by looking at the path registered in the packet
- Example :

<u>How **u** learns a path towards **z** :</u>

1) **z** sends a HELLO packet to all his neighbors

2) Each node receiving a HELLO packet pushes its node identifier (or its edge identifiers) into the packet's path trace

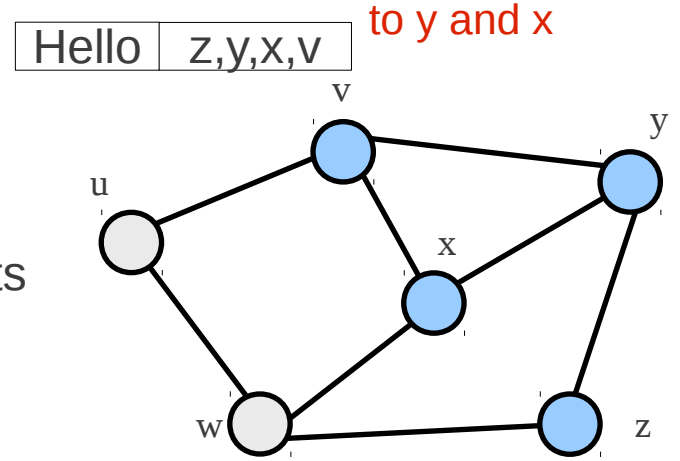
3) When **u** receives a HELLO packet (originated from **z** or relayed by **z**) it can retrieve a path to **z** by inverting the packet trace



Path-vector protocols (3/4)

- Main idea :
  - Discovery packets traveling around the network, keep a record of the traversed nodes (or edges)
  - Paths to a resource of interest can therefore be simply obtained by looking at the path registered in the packet

In this case, u also learns to y and x



How **u** learns a path towards **z** :

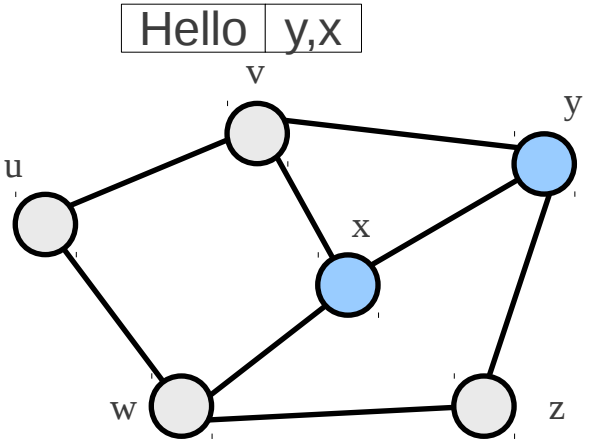
1) **z** sends a HELLO packet to all his neighbors

2) Each node receiving a HELLO packet pushes its node identifier (or its edge identifiers) into the packet's path trace

3) When **u** receives a HELLO packet (originated from **z** or relayed by **z**) it can retrieve a path to **z** by inverting the packet trace

Path-vector protocols (4/4)

- Main idea :
  - Discovery packets traveling around the network, keep a record of the traversed nodes (or edges)
  - Paths to a resource of interest can therefore be simply obtained by looking at the path registered in the packet
- Example :



<u>How to avoid having the HELLO messages looping forever :</u>

• Before relaying a HELLO message to a neighbor node, the relay node checks that the neighbor isn't contained in the packet's path history

- Reason : If it is included, relaying to the neighbor would create a loop

• Once a packet has traversed all nodes it will no longer be relayed

Distance Vector Protocols (1/5)

General operation:

- Each node receives distance information from its neighbors to different nodes in the network

  E.g. from v: D(v, z) = 2

  D(v, x) = 3

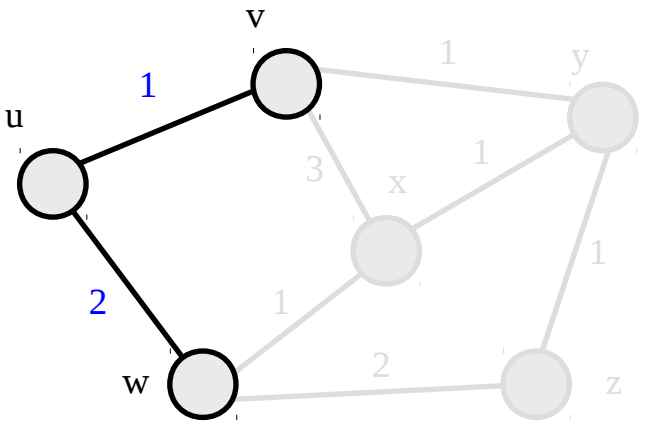
  from w: D(w, z) = 2

- Computes route distances from itself to all other nodes in the network using the Bellman-Ford algorithm

  D(i,k) = min { cost(i,j) + D(j,k) }

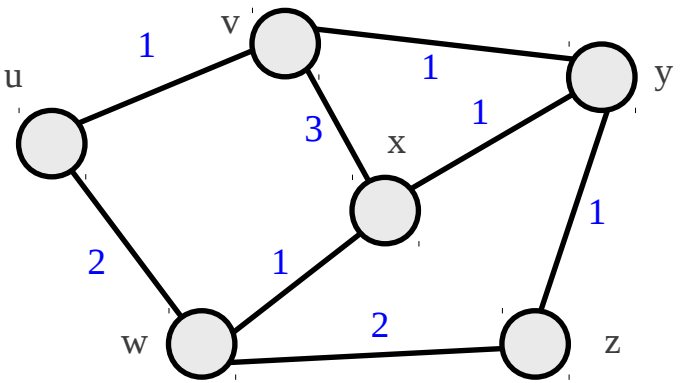
  E.g. D(u, z) = min {2+1, 2+2} = 3

- Updates its distance vectors and sends updates to neighbors



Protocol converges when there is no updated information to be exchanged

Distance Vector Protocols (2/5)



Sample RIP step

Distance vectors

Route update

u

	u	v	w	x	y	z
u	0	1	2	∞	∞	∞
v	∞	∞	∞	∞	∞	∞
w	∞	∞	∞	∞	∞	∞

v

	u	v	w	x	y	z
u	∞	∞	∞	∞	∞	∞
v	1	0	∞	3	1	∞
x	∞	∞	∞	∞	∞	∞
y	∞	∞	∞	∞	∞	∞

w

	u	v	w	x	y	z
u	∞	∞	∞	∞	∞	∞
w	2	∞	0	1	∞	2
x	∞	∞	∞	∞	∞	∞
z	∞	∞	∞	∞	∞	∞

	u	v	w	x	y	z
u	0	1	2	3	2	4
v	1	0	∞	3	1	∞
w	2	∞	0	1	∞	2

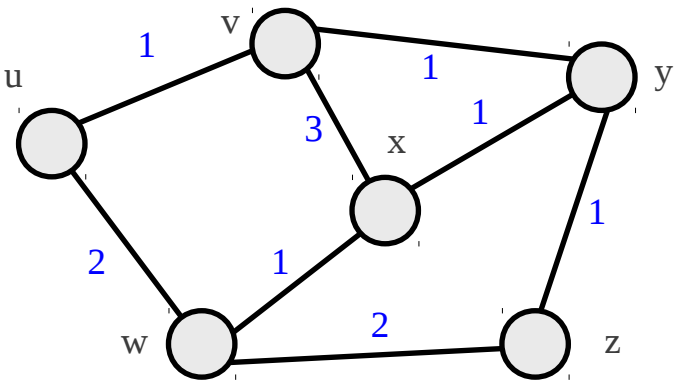
	u	v	w	x	y	z
u	0	1	2	∞	∞	∞
v	1	0	3	2	1	2
x	∞	3	1	0	1	∞
y	∞	1	∞	1	0	1

	u	v	w	x	y	z
u	0	1	2	∞	∞	∞
w	2	3	0	1	2	2
x	∞	3	1	0	1	∞
z	∞	∞	2	∞	1	0

etc …

Distance Vector Protocols (3/5)

Sample RIP step



Distance vectors

u

	u	v	w	x	y	z
u	0	1	2	3	2	3
v	1	0	3	2	1	2
w	2	3	0	1	2	2

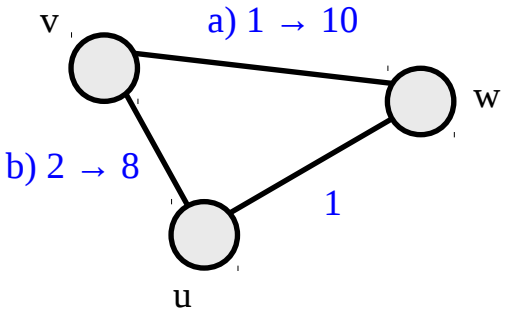
- At the end, each node will know his shortest graph distance and the distance of his neighbors to every other node in the network

- Selecting the next hop to a destination is then fairly simple : pick the neighbor that is closest to the destination

Distance Vector Protocols (4/5)

Problem with distance vectors: “count to inf”

Imagine change (a) takes place, then w advertises to u, then change (b) takes place



Destination

DV from

	u	v	w
u	0	2	1
v	2	0	1
w	1	1	0

w’s state BEFORE link change (a)

	u	v	w
u	0	2	1
v	2	0	1
w	1	**3**	0

link change detected by w which wrongly assumes that v is reachable via u at distance 2+1

	u	v	w
u	0	**4**	1
v	2	0	1
w	1	**3**	0

the mistake is advertised to u which after change (b) thinks it can best reach v via w at dist 3+1

	u	v	w
u	0	**4**	1
v	8	0	9
w	1	**3**	0

v receives advertisements from u and w but is fine

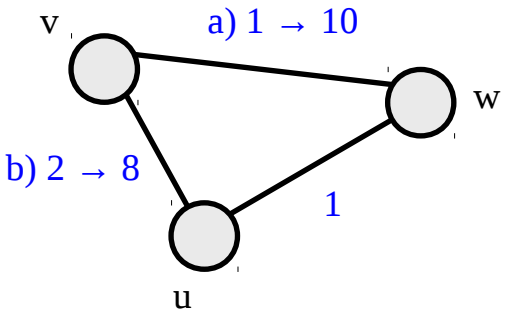
	u	v	w
u	0	**4**	1
v	8	0	9
w	1	**5**	0

when u advertises to w the mistake comes back to w that now thinks it can see v via u at dist 4+1

Distance Vector Protocols (5/5)

Solution for “count to inf”: Poison reverse

w: send infinity for destination v to neighbor u if u is the next hop to y



Destination

DV from

	u	v	w
u	0	2	1
v	2	0	1
w	1	1	0

w’s state BEFORE link change (a)

	u	v	w
u	0	2	1
v	2	0	1
w	1	**3**	0

W sends to u and infinite Distance towards v

	u	v	w
u	0	8	1
v	2	0	1
w	1	**3**	0

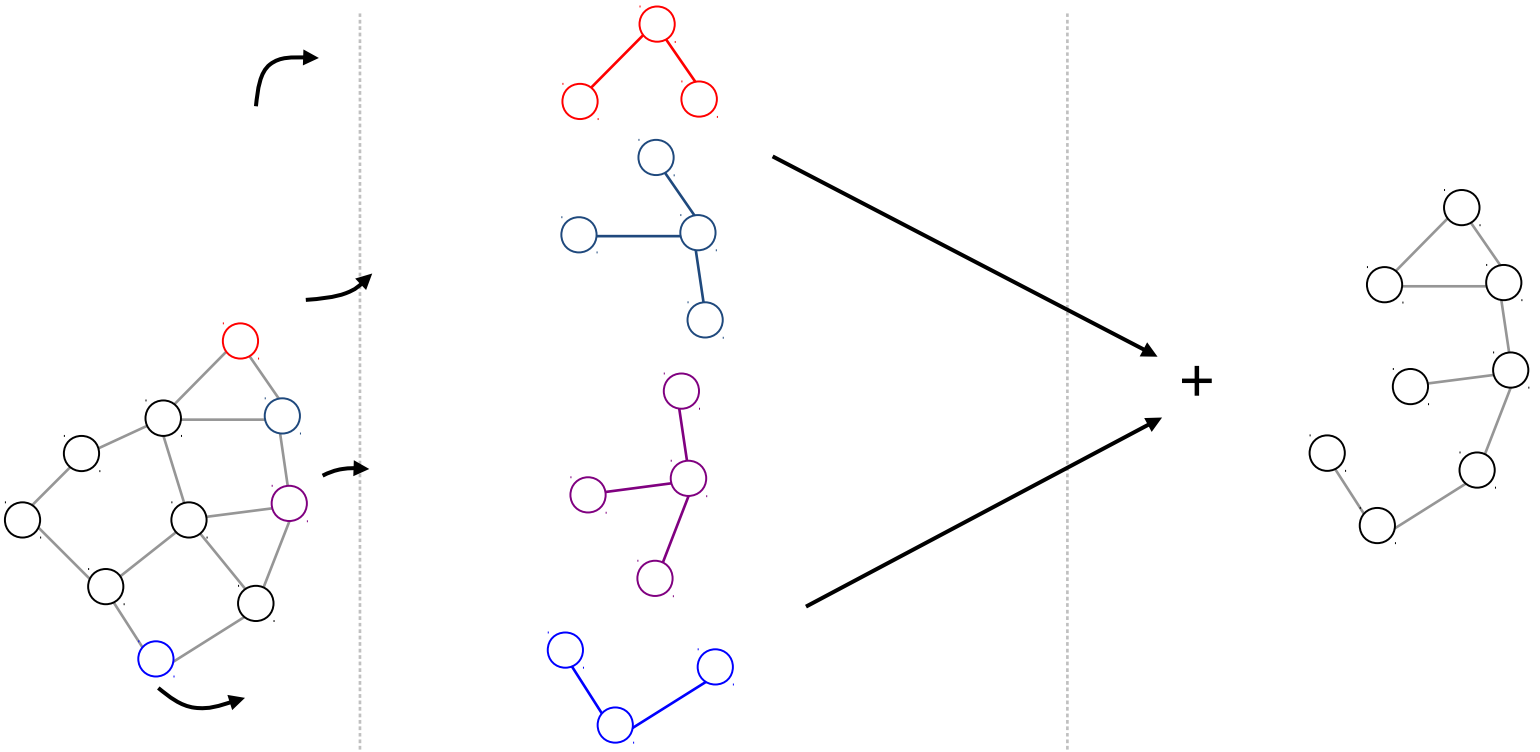
	u	v	w
u	0	**8**	1
v	8	0	9
w	1	**9**	0

v receives advertisements from u and w but is fine

Unfortunately, poison reverse does not solve all the cases of the count-to-infinity problem.

Link-state protocols (1/3)

General operation



Topology

Each router monitors the state of its direct neighborhood (HELLO protocol) and if there is a topology change floods this information to entire network (Flooding protocol)

Each router incrementally builds the entire topology as it learns the link state of other routers. Then it periodically computes the paths using a shortest path algorithm

Link-state protocols (2/3)

Route computation: Dijkstra's SPF algorithm

Can be implemented with O(m long n) complexity for m links and n nodes

E: evaluated nodes, R: remaining nodes, O: ordered list of paths

1. Init: E = {u}, R = {v,w,x,y,z}, O = {uv=1, uw=2}
2. if O == {Ø} or infinite distances, terminate
3. let P the shortest path in O remove P from O
let V last node in P

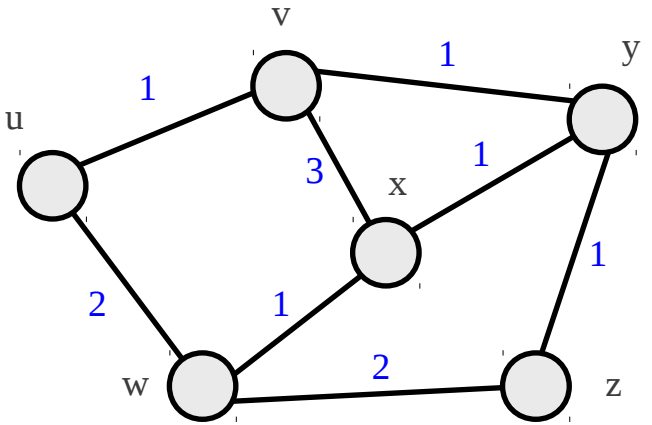
    if V in E goto 2

    else

        mark P shortest path to V

        move V from R to E

4. add to O new paths by appending to P all links to V's neighbors
goto step 2

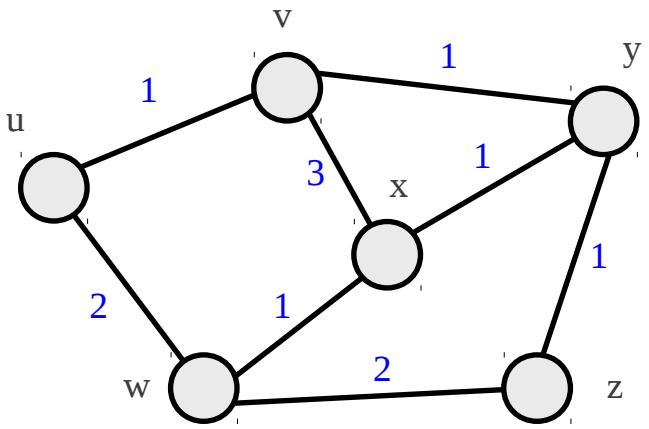


NOTE: At this point, any other shortest path algorithm may be used (even Bellman-Ford), or a BFS traversal if all edges have the same weight

Link-state protocols (3/3)

Step	E	R	O	SP List
0	u	vwxyz	uv=1 uw=2	uv=1
1	uv	wxyz	uw=2 uvy=2 uvx=4	uv=1 uw=2
2	uvw	xyz	uvy=2 uvx=4 uwx=3 uwz=4	uv=1 uw=2 uvy=2
3	uvwx	yz	uvx=4 uwx=3 uwz=4 uvyz=3 uvyx=3	uv=1 uw=2 uvy=2 uwx=3
4	uvwxy	z	uvx=4 uwz=4 uvyz=3 uvyx=3 uwxy=4 uwxv=6	uv=1 uw=2 uvy=2 uwx=3 uvyz=3
5	uvwxyz			

Sample run for node *u*

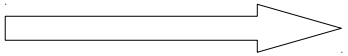


Comparison of discovery methods

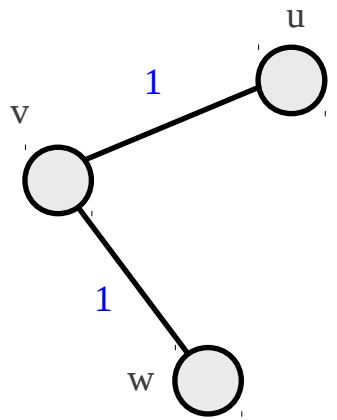
- Link state operates based on global topology knowledge while DV and PV have a partial view of the network
- Link state and path-vector methods allow to see the links existing in the network. In DV, we simply collect distances between neighbors and not their relations

If the edges of the graph are undirected and unweighed (or equally weighed), the edges can be induced between nodes having the minimum distance between each other

	u	**v**	**w**
u	0	1	2
v	1	0	1
w	2	1	0



we can infer



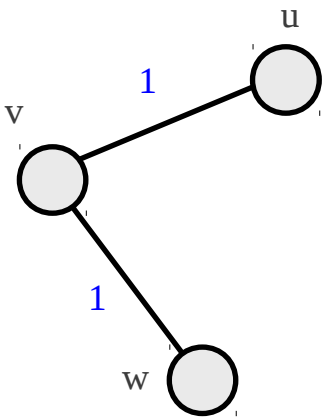
Comparison of discovery methods

- Link state operates based on global topology knowledge while DV and PV have a partial view of the network
- Link state and path-vector methods allow to see the links existing in the network. In DV, we simply collect distances between neighbors and not their relations

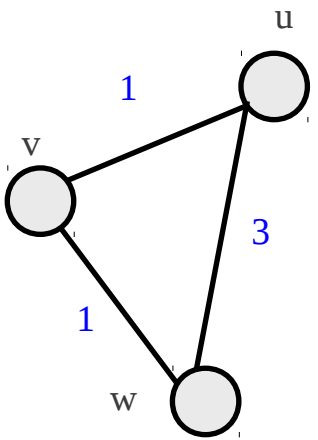
But if the edges of the graph can be weighted, the distance matrix alone cannot allow to infer all edges

	u	v	w
u	0	1	2
v	1	0	1
w	2	1	0

Can correspond to



Or also



This reduces the flexibility to choose paths other than the shortest path when using DV

What if multiple paths are possible ?

- In case multiple paths to a destination are possible, the routing process needs to select the *best* path before pushing it to forwarding
- This notion of *best* may depend on many factors :
  - Different metrics :

    number of hops, bandwidth, latency, cost, link quality (data rate, error rate, interference), Traffic / congestion index …

  - Local node preferences (usually human input): also called policies

Scalability

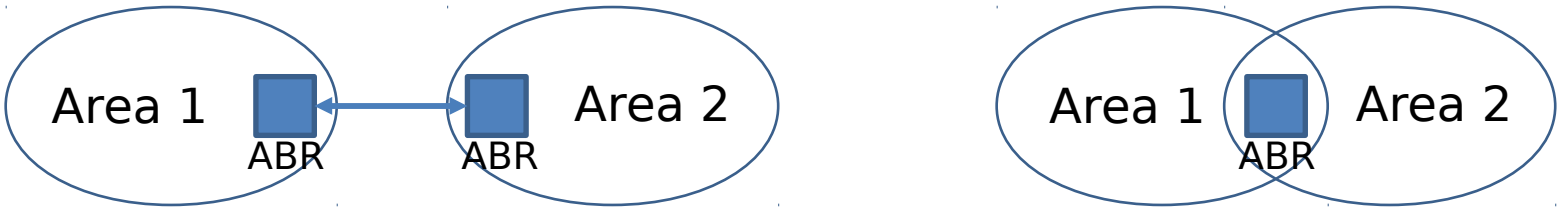
Route Discovery protocols heavily rely on group communications :

this does not scale well with large networks

Solution :

consider clustering of the network into *Areas* that operate as independent routing domains

Across Areas, *area border routers* (ABR) summarise routing information and distributed it (subject to policy) into other Areas



Outcome of the discovery phase

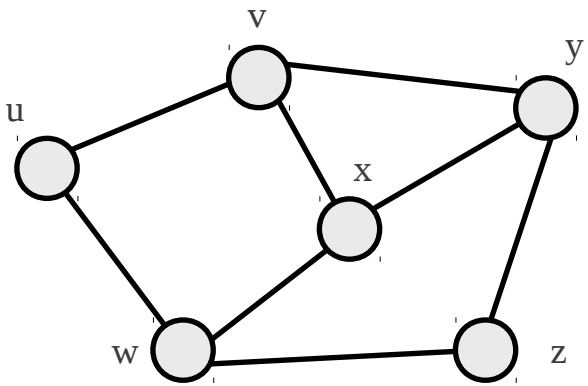
RIB : Routing Information Base

Name or Address	Path
Node 1	n4, n3, n2, n5, n1
Node 2	n4, n2
…	...

- These paths need to be transmitted to forwarding
- Can be done in 2 ways :
  - Source routing
  - Forwarding Information Base

Push path into packet : Source routing

- The computed paths of the RIB can be indicated to forwarding by storing the computed path into the sent packet.



Message from **u** to **z** :

1) **u** sends packet with the chosen path in the header

u,v,y,z	Payload

2) transit nodes **v** and **y** forward to the neighbor indicated after their position in the path

3) **z** can respond to **u** by simply sending a packet along the reverse path

z,y,v,u	Payload

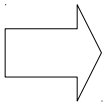
- This practice is called **source routing**. Advantages :
  - The source has full control on the path taken by its packets
  - Only the source needs to perform routing, other nodes simply do forwarding

Push the path into a forwarding table

- If each node maintains a forwarding table, there is no need to store the full path for forwarding, only the next hop will do !
  - In the example below, **Node 4** will have an entry in its forwarding table for **Node 1** indicating **Node 3** as a next hop.

Name or Address	Path
Node 1	n4, n3, n2, n5, n1
Node 2	n4, n2
…	...

RIB



Name or Address	Next hop
Node 1	n4
Node 2	n4
…	...

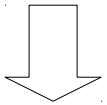
FIB : Forwarding Information Base

- What about the possible size of such a table ? Small calculation : IPv4 addresses are exhausted which means there are 2^{32} = 4.29 billion addresses we could forward to, I.e as many entries
  - Such a large list slows down forwarding
  - It might not fit in every network node (router) out there : around 16 GB for the first column alone

Aggregating the FIB

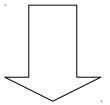
- If the addresses or names have a structure allowing to do so, one can aggregate the entries of the table, examples :

Name	Next hop
1	4
2	4
3	4
…	...



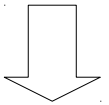
Name	Next hop
1 to 3	4
…	...

Address	Next hop
10.2.3.4	4
10.2.3.1	4
10.2.1.1	4
…	...



Address	Next hop
10.2.0.0	4
…	...

Name	N.h
informatik.unibas.ch	4
cs.unibas.ch	4
urz.unibas.ch	4
…	...

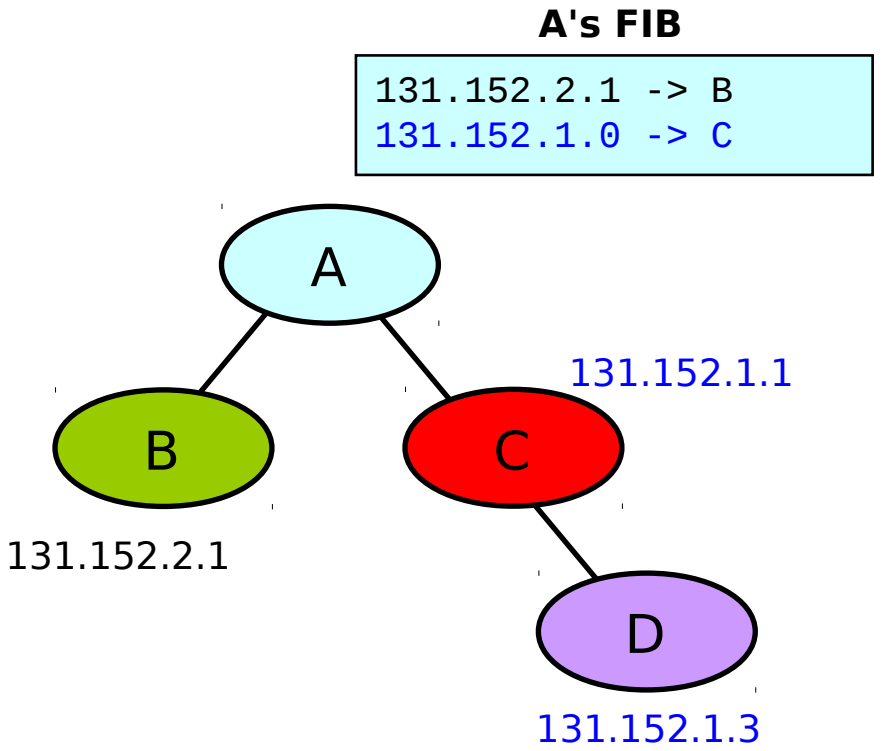


Name	N.h
*.unibas.ch	4
…	...

Aggregating the FIB

Main condition for aggregation to be possible : the candidates for aggregation need to share the same next hop.

Said differently it means that addresses (or names) that are similar to each other should belong to nodes that are close to each other in the network.



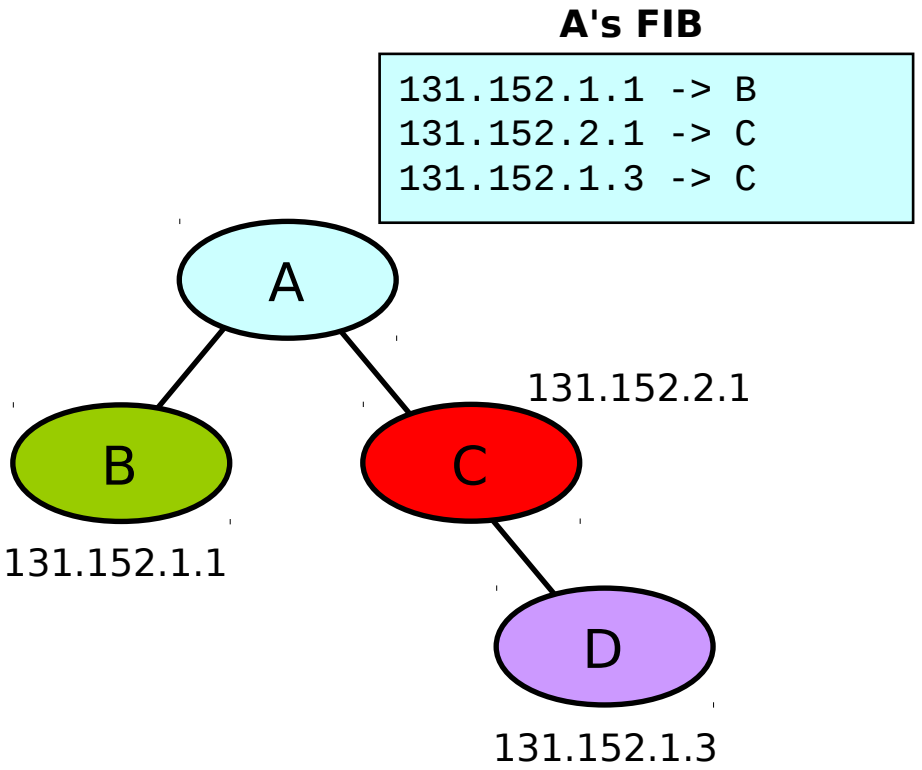
In this case, addresses are attributed coherently since nodes **C** and **D** have similar addresses and are close to each other.

Hence **A**, can compact the entries for these 2 nodes in a single one.

Aggregating the FIB

Main condition for aggregation to be possible : the candidates for aggregation need to share the same next hop.

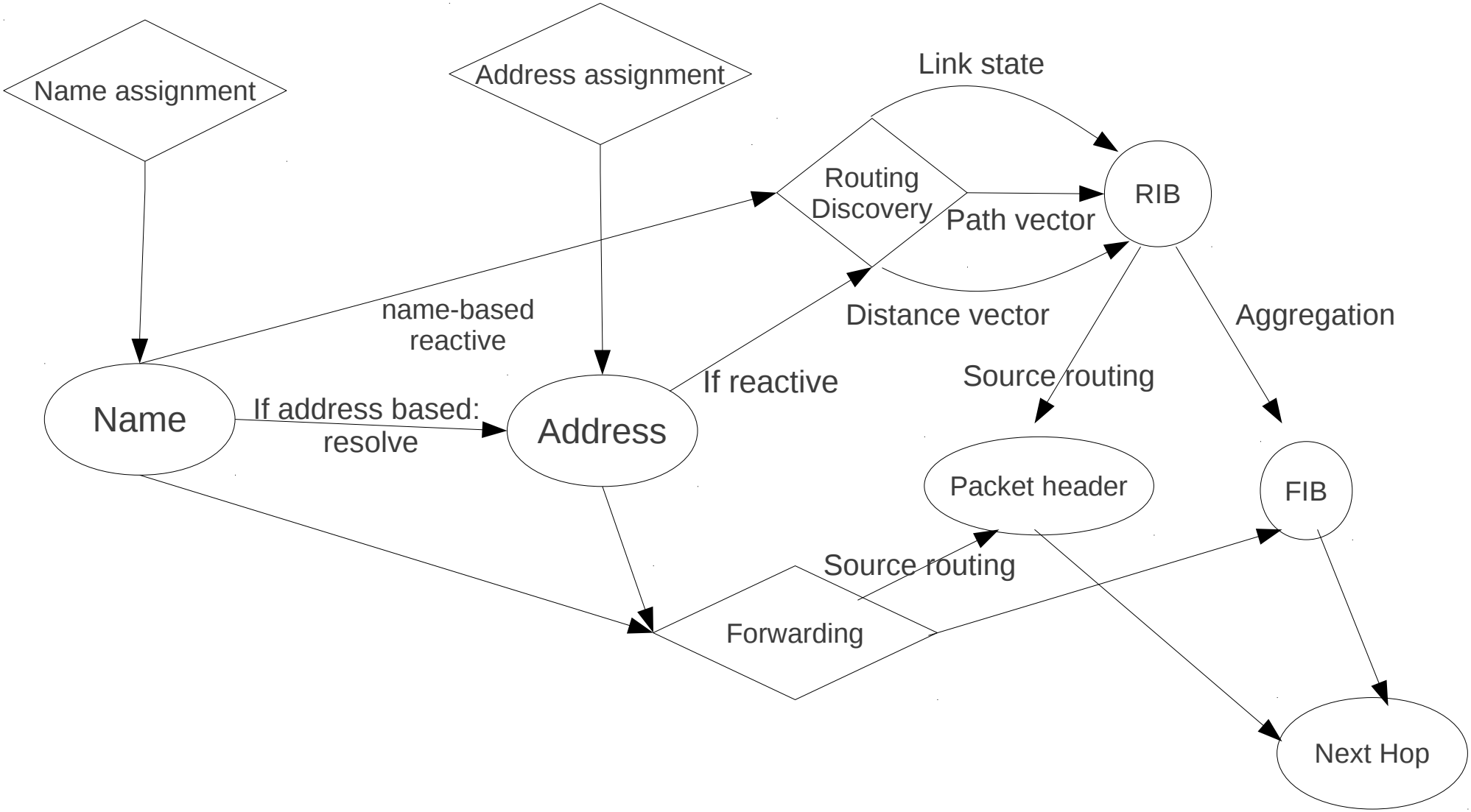
Said differently it means that addresses (or names) that are similar to each other should belong to nodes that are close to each other in the network.



In this case, aggregation is not possible since addresses do not reflect the positions in the network.

This problem exists in current Internet and is known as the **IP address fragmentation problem**

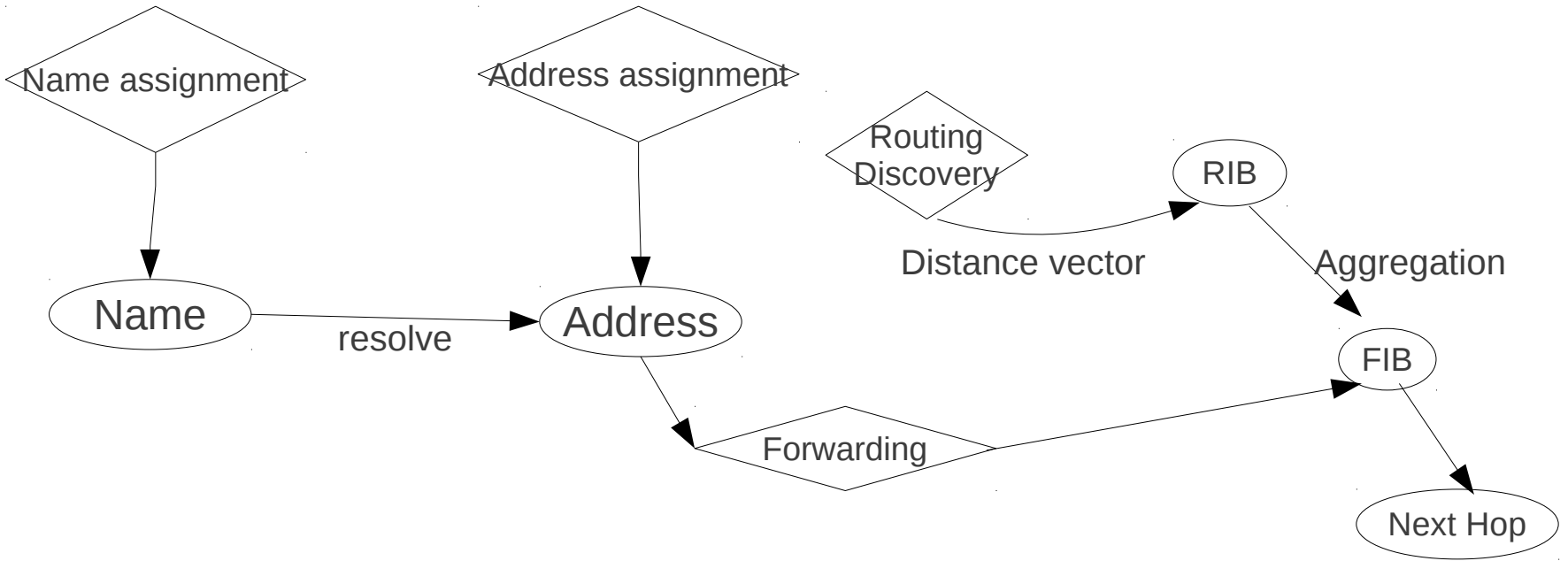
Overview


Name assignment
Address assignment
Link state
Routing Discovery
RIB
Path vector
name-based reactive
Distance vector
Aggregation
Source routing
Name
If address based: resolve
Address
If reactive
Packet header
FIB
Source routing
Forwarding
Next Hop

Example protocols : Routing Information Protocol (RIP)

- RIP is a distance-vector protocol
- Vector information updated every time a node receives new distance information

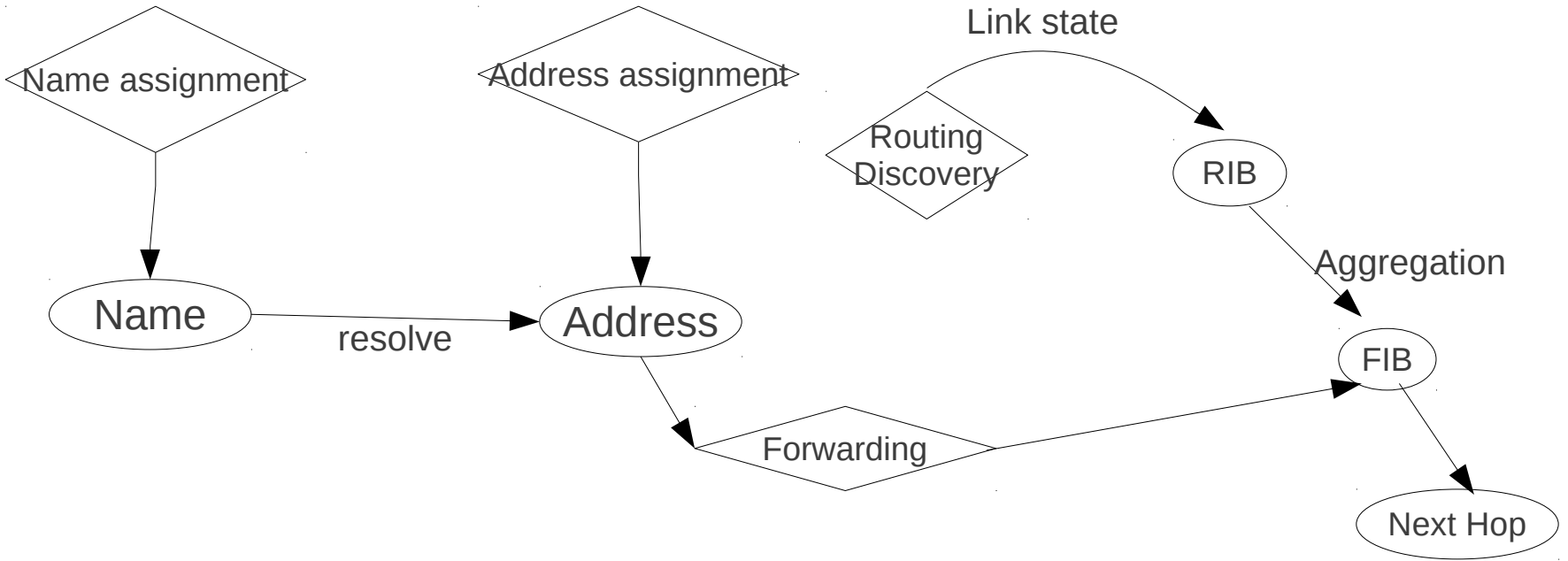
One of the first routing protocol used in the Internet, and the oldest still in use!
Very simple, asynchronous, hop-by-hop computation
Not very scalable!



Example protocols : Open Shortest Path First Protocol (OSPF)

- OSPF is a link-state protocol used in large wired networks
- The OSPF protocol has 3 sub-protocols
  - Link Database Exchange protocol : ensures reliable transfer
  - HELLO protocol
  - Topology change Flooding protocol

It is highly complex ... more than it seems in these slides!



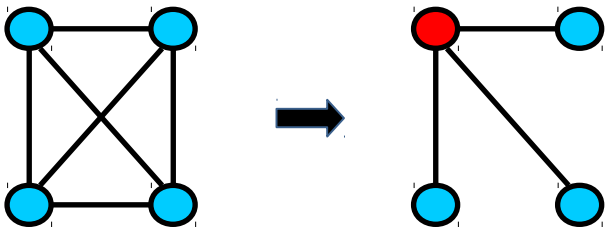
Example protocols : Open Shortest Path First Protocol (OSPF)

- OSPF is a link-state protocol used in large wired networks
- The OSPF protocol has 3 sub-protocols
  - Link Database Exchange protocol : ensures reliable transfer
  - HELLO protocol
  - Topology change Flooding protocol

It is highly complex ... more than it seems in these slides!

HELLO protocol used to elect a *designated router* and a *backup designated router* in broadcast mediums in order to reduce the number of broadcasted messages

- Everyone exchanges information with the designated router only

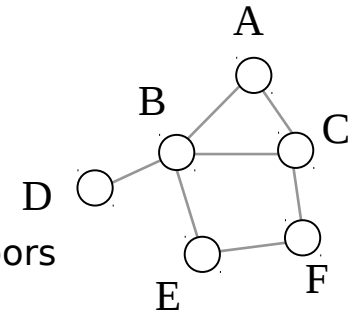


Optimised Link-State Routing Protocol (1/4)

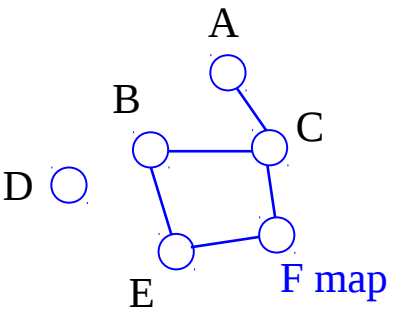
2-hop neighbourhood discovery phase

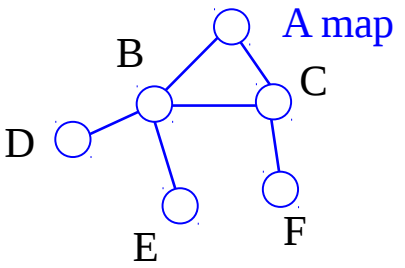
Every node broadcasts to its immediate neighbours HELLO messages informing of its 1-hop peers. E.g.:

- A broadcasts HELLO_A{Ø} (no neighbors) -> B and C learn 1-hop neighbor A
- B broadcasts HELLO_B{A} -> A, C, D, and E learn 1-hop neighbor B
- C, D, and E learn that B has neighbor A
- C broadcasts HELLO_C{A} -> A, B, F learn they have 1-hop neighbor C
- B, F, learn that C has neighbor A
- A broadcasts now HELLO_A{B, C} -> B and C learn they are 2-hop neighbors via A
- D broadcasts HELLO_D{B} -> B learns it has 1-hop neighbor D
- B broadcasts now HELLO_B{A,C,D} -> A, C, E learn 2-hop neighbor D through B
- A, D, and E learn 2-hop neighbor C through B
- E broadcasts HELLO_E{B} -> F learns it has neighbor E and 2-hop neighbor B
- …



After the protocol converges (all nodes establish consistent maps) all nodes know their 1 and 2-hop neighbors



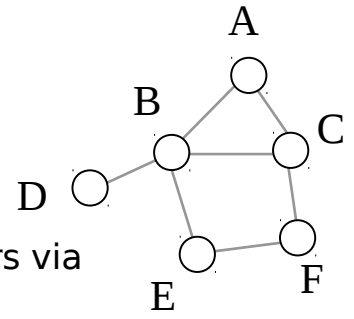


Optimised Link-State Routing Protocol (2/4)

2-hop neighbourhood discovery phase

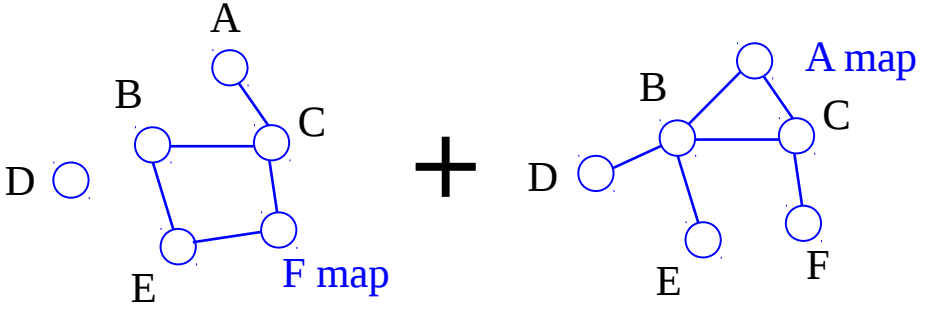
Every node broadcasts to its immediate neighbours HELLO messages informing of its 1-hop peers. E.g.:

- A broadcasts HELLO_A{Ø} (no neighbors) -> B and C learn 1-hop neighbor A
- B broadcasts HELLO_B{A} -> A, C, D, and E learn 1-hop neighbor B
- C, D, and E learn that B has neighbor A
- C broadcasts HELLO_C{A} -> A, B, F learn they have 1-hop neighbor C
- B, F, learn that C has neighbor A
- A broadcasts now HELLO_A{B, C} -> B and C learn they are 2-hop neighbors via A
- D broadcasts HELLO_D{B} -> B learns it has 1-hop neighbor D
- B broadcasts now HELLO_B{A,C,D} -> A, C, E learn 2-hop neighbor D through B
- A, D, and E learn 2-hop neighbor C through B
- E broadcasts HELLO_E{B} -> F learns it has neighbor E and 2-hop neighbor B
- ...



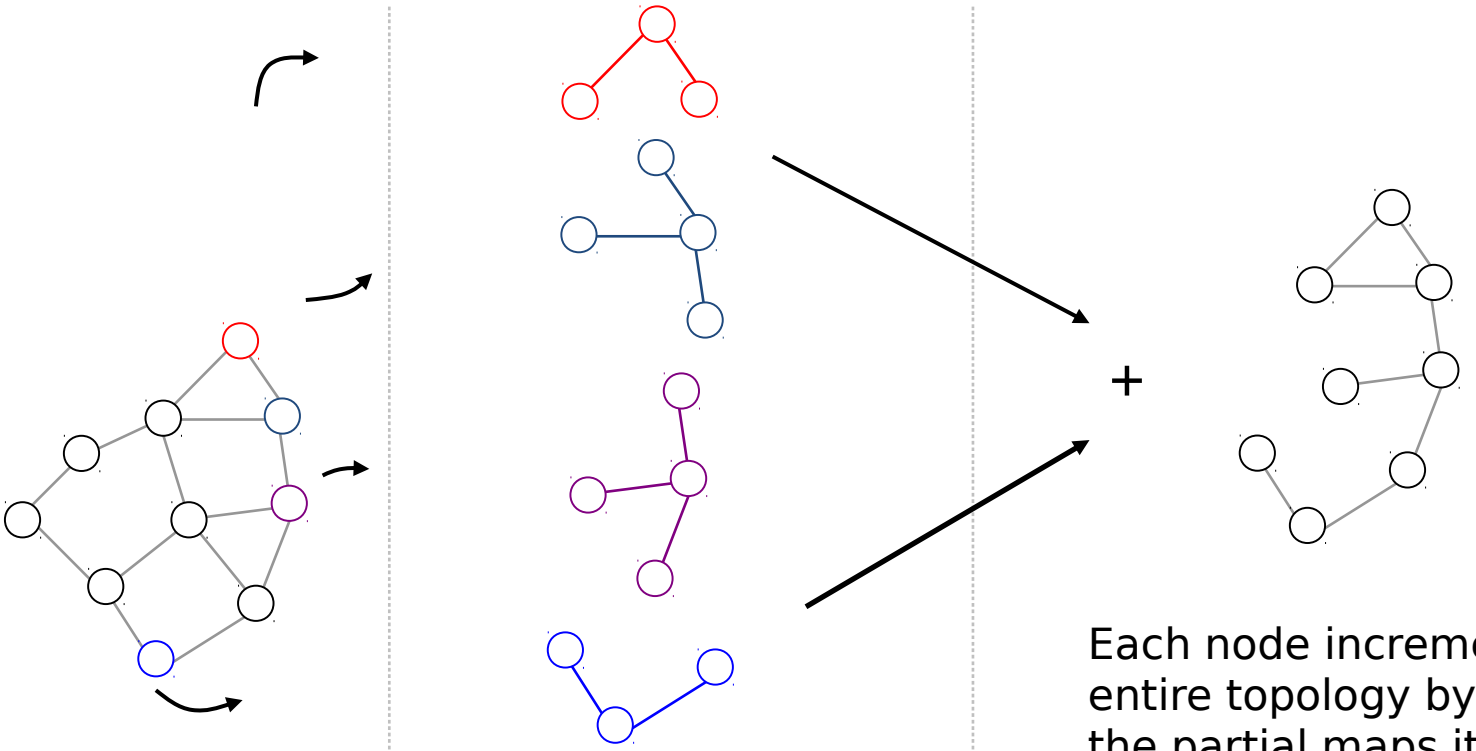
After the protocol converges (all nodes establish consistent maps) all nodes know their 1 and 2-hop neighbors

Consistent maps does not mean the same maps though, but rather partial, non-conflicting and complementary



Optimised Link-State Routing Protocol (3/4)

B. Topology advertisement phase



Topology

Each node advertises its partial map to the entire network using "flooding"

Each node incrementally builds the entire topology by superimposing the partial maps it receives (link state information)

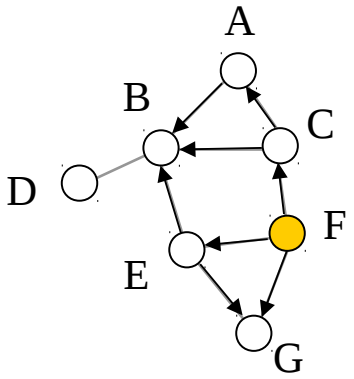
Then run a path computation algorithm (e.g. Dijkstra) to decide the paths (and build forwarding tables).

Optimised Link-State Routing Protocol (4/4)

Differences to OSPF:

- OLSR selects Multi-Point Relay nodes and relies on them to do the topology announcement broadcasts
- Members of the MPR list are the only nodes allowed to relay messages in the flooding phase !
  - This is sufficient since all nodes in the topology are 1-hop away from the CDS (last lecture)
  - Significantly reduces the control traffic generated by the routing protocol !!!
- OLSR does not use a reliable protocol for transferring link information since wireless networks are unreliable anyway. It just floods topology data often enough instead.

Ad-hoc On-demand Distance Vector Protocol



RREQ

<u>Prepare reverse paths first</u>

F floods a route request (RREQ) packet in the entire network to indicate it is "looking for B". RREQ has a hop counter set to 0.

All intermediate nodes receiving the RREQ

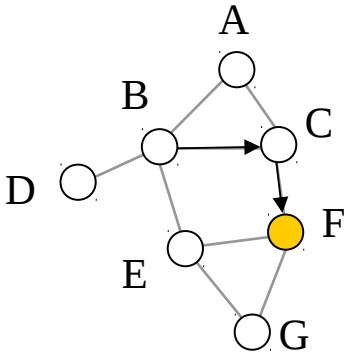
- Increase hop counter and re-broadcast the message the first time they receive it
- Create temporary routing state to reach back node F

<u>Activation of forward path</u>

When B receives the RREQ (possibly multiple copies),

- Logs from which neighbor it received a RREQ with the lowest hop-count (shortest path)
- sends a route reply (RREP) message back to F that activates the routing state in intermediate nodes (here node C)

F can now send data to node B



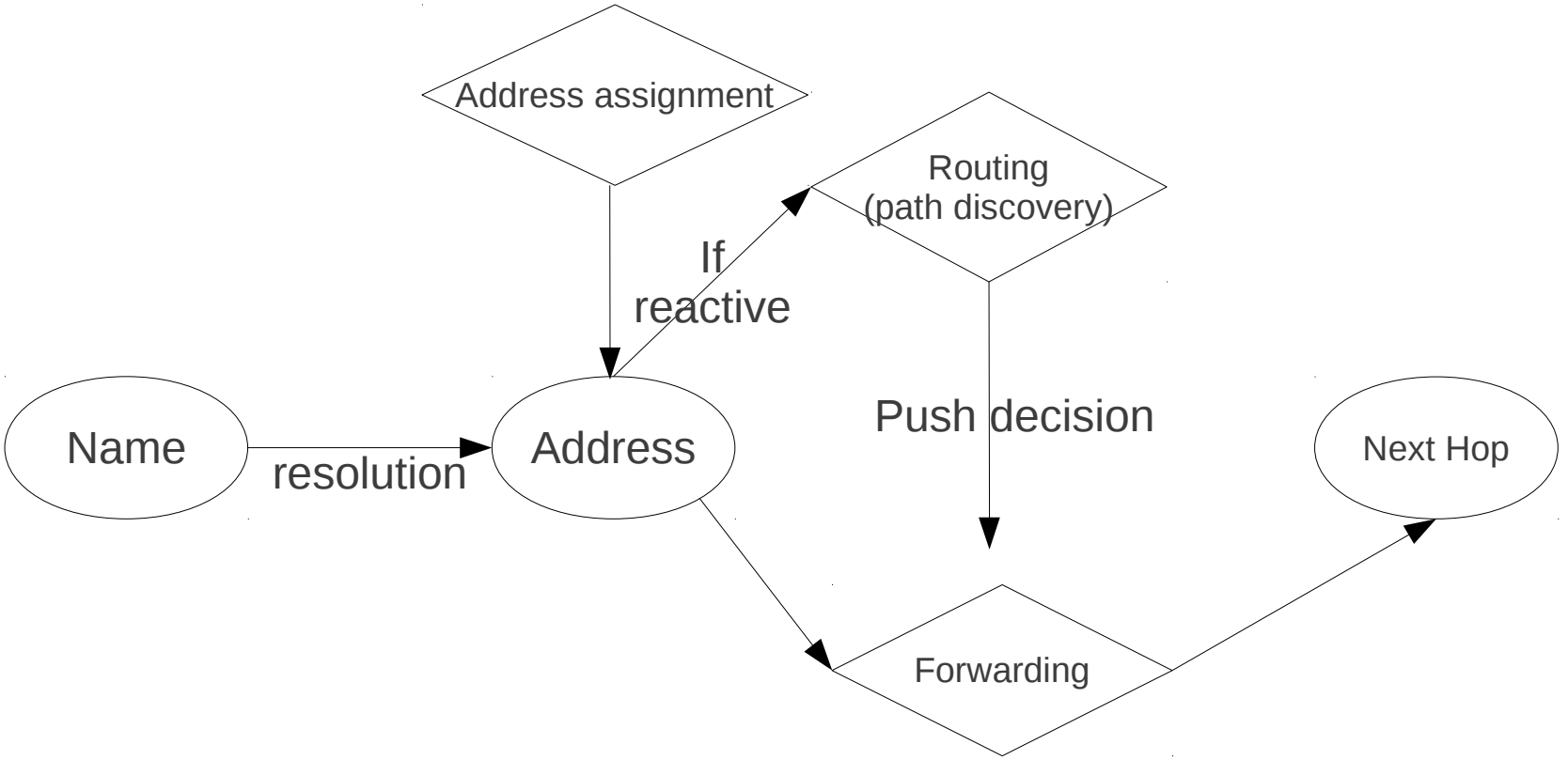
RREP

Intermediate nodes also learn and activate the routes to B and F

Intermediate nodes that already have a route to some target destination can reply and stop propagating the RREQ message

Greedy Routing

Location and addresses



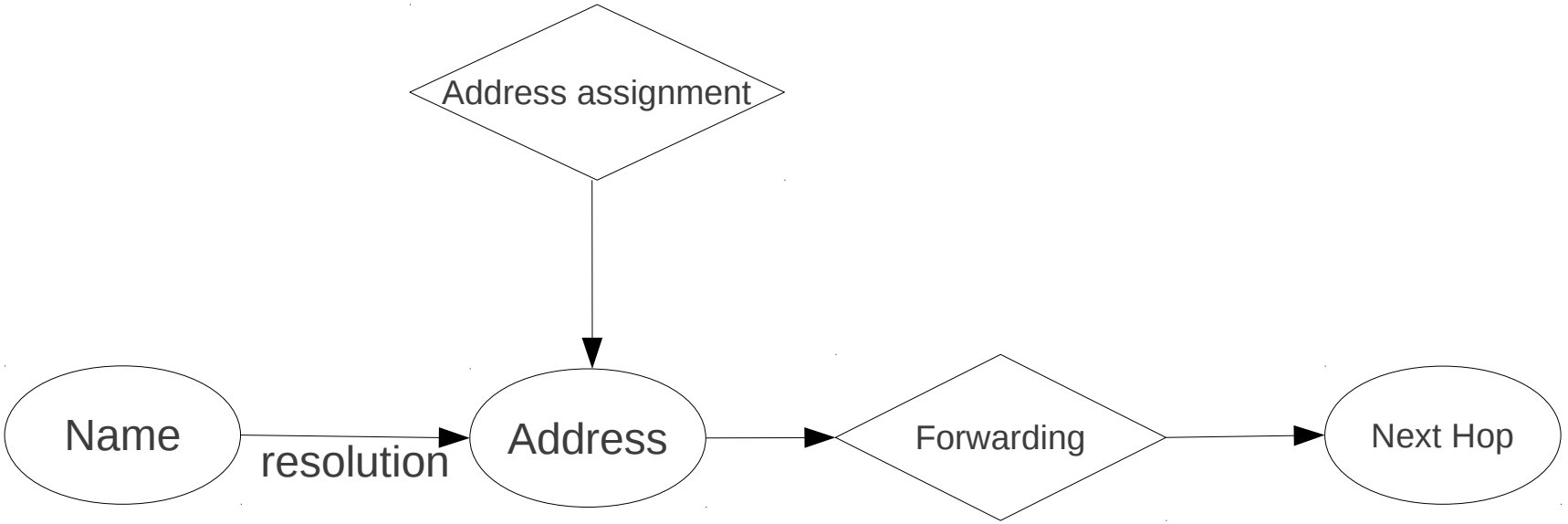
Question : If addresses really do indicate a location information, then why did we still require a discovery phase ?

In other words if addresses are assigned correctly, we shouldn't require any routing !

Misconception of locations and addresses

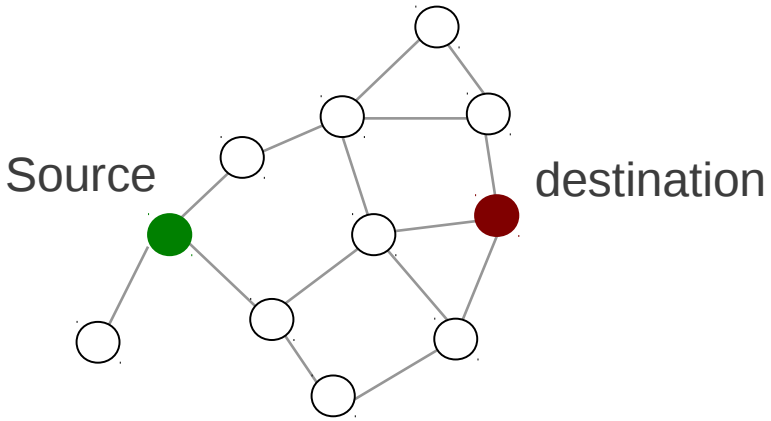
- An address should indicate the location of the resource
- An address in our common use is : Bernoullistrasse 16, Basel, Switzerland
  - A combination of names
  - Is this really a “locator” ?
  - To you maybe yes, since you live in Basel and you already did “the routing” to map this collection of names to a location and hence to a path
  - Would you be able to indicate me a path from 17 Place de l'Etoile to 12 Place st-Etienne in Strasbourg ?
- If the addresses we used contained location information you should be able to do so : example GPS coordinates !
  - If I give you a source coordinate and a destination one (and tell which direction I am looking), you can always give me a path (direction) and even an estimation of the time it will take

Location and addresses



How would forwarding function if the addresses correctly reflected network proximity ?

Greedy routing



1) The source is assumed to know its location (address) **s** and the location of its neighbors

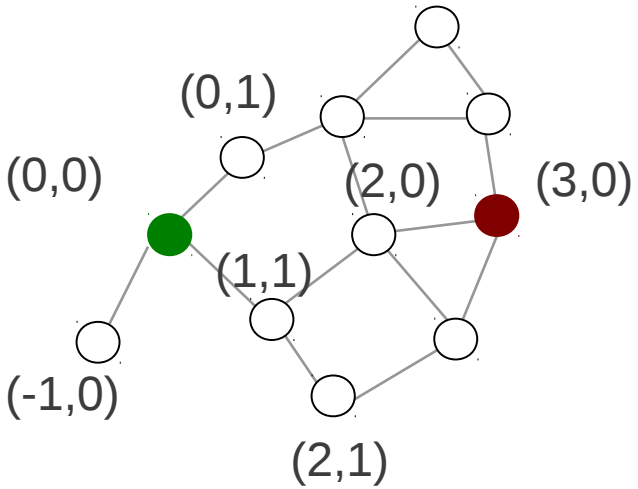
2) A distance function **D** is assumed to estimate the proximity of 2 given addresses

When Sending from source to destination

1) the source calculates the distance between the addresses of all its neighbors and the destination

2) forwards the message to the neighbor with an address closest to destination

Greedy routing – example in Euclidean space



In this case the addresses assigned are coordinates in the 2-D plane and the distance function between them is the Euclidean distance.

Source wants to send from its location (0,0) to the destination's location (3,0)

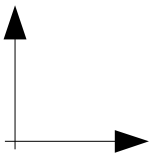
The message follows the path :

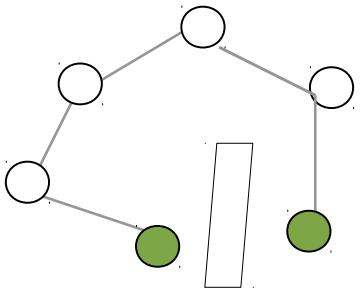
(0,0) → (1,1) → (2,0) → (3,0)

How can we find such nice coordinates ?

Why not GPS (or real world) coordinates ?

- There is a correlation between real-world position and network location :
  - Nodes in the same geographic region a more likely to have similar network connectivity
  - This is not always true :
    - Example :
      - 2 neighbors in the same building connected to 2 different service providers
      - Or in wireless networks, 2 nodes at the same location that cannot communicate due to an obstacle





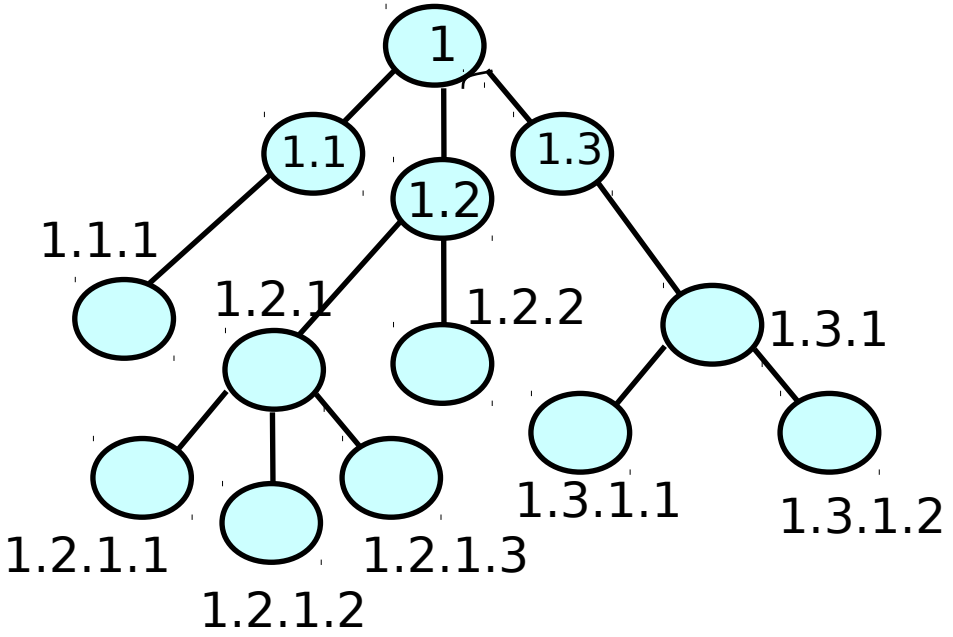
Obstacle blocking radio signal

These 2 nodes are physically close but at opposite locations in the network graph.

Therefore forwarding based on physical locations can lead into errors

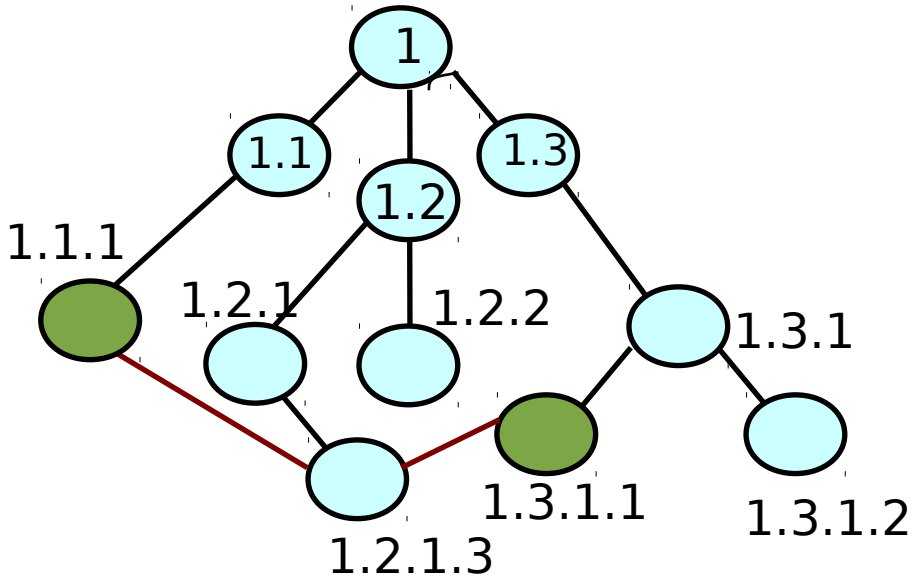
Why not hierarchical addresses ?

• When assigned correctly (without fragmentation), hierarchical addresses do reflect a notion of locality in the graph and hence the possibility of greedy routing



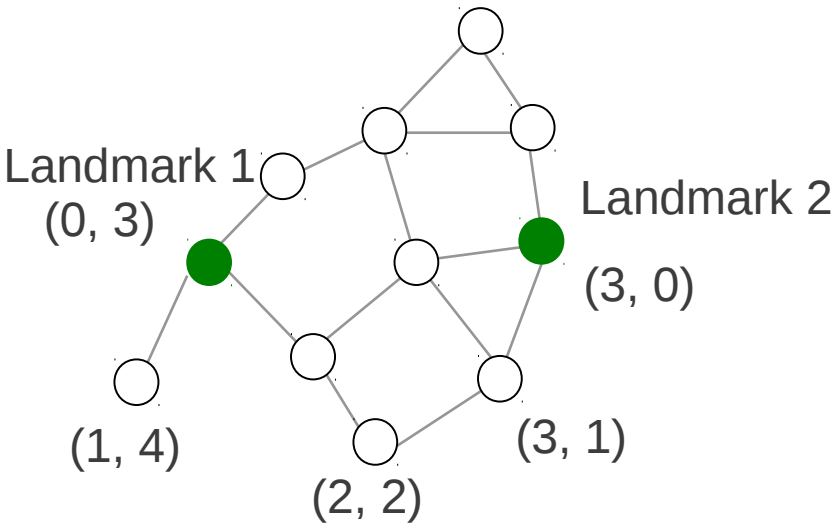
Why not hierarchical addresses ?

- Problem : Most of the network graphs are not trees



These 2 nodes are close to each other in the network, however their addresses do not indicate so !

Landmark coordinates



1) a few nodes are selected to act as landmarks

2) each node in the network finds out its shortest distance to all landmarks

3) the coordinates of a node is its distance vector to the landmarks (landmarks have distance 0 to themselves)

4) different distance functions could be applied between the 2 vectors : euclidean, Manhattan …

5) simplest forwarding strategy : first go towards the landmark closest to destination

Disadvantages :

- Strongly depends on the placement of the landmarks
- Many landmarks required for large networks

Questions ?