Internet of Things

5.3 Mesh Networking

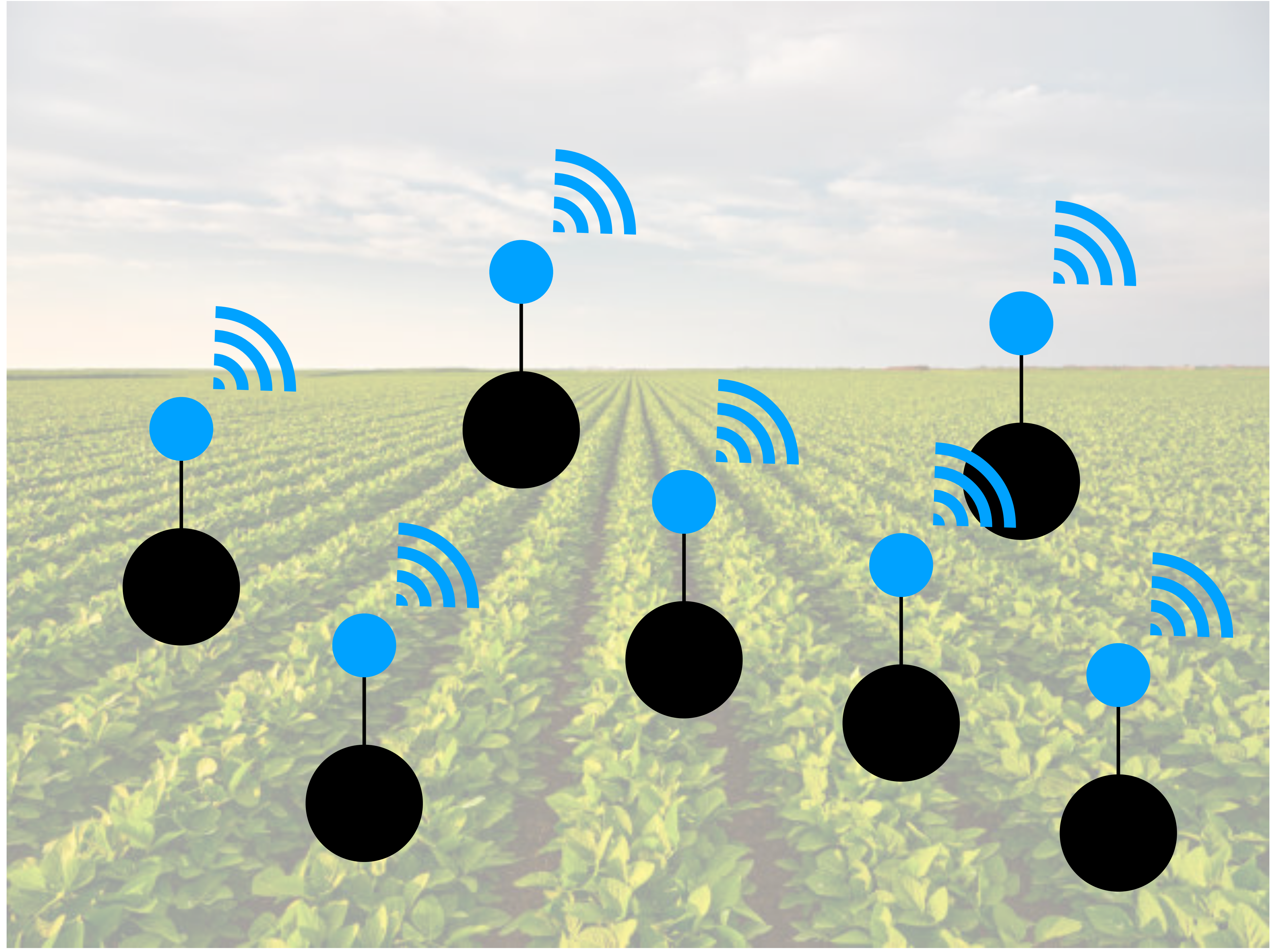
M.Sc. Computer Science 2024-2025

Viviana Arrigoni

Motivation - IoT environments

Motivation - IoT environments

- Distribute a bunch of nodes in some environment
- Need to collect data, monitor
- No reliable wireless infrastructure
- They need to cooperate and work together to achieve tasks



Mesh Networking

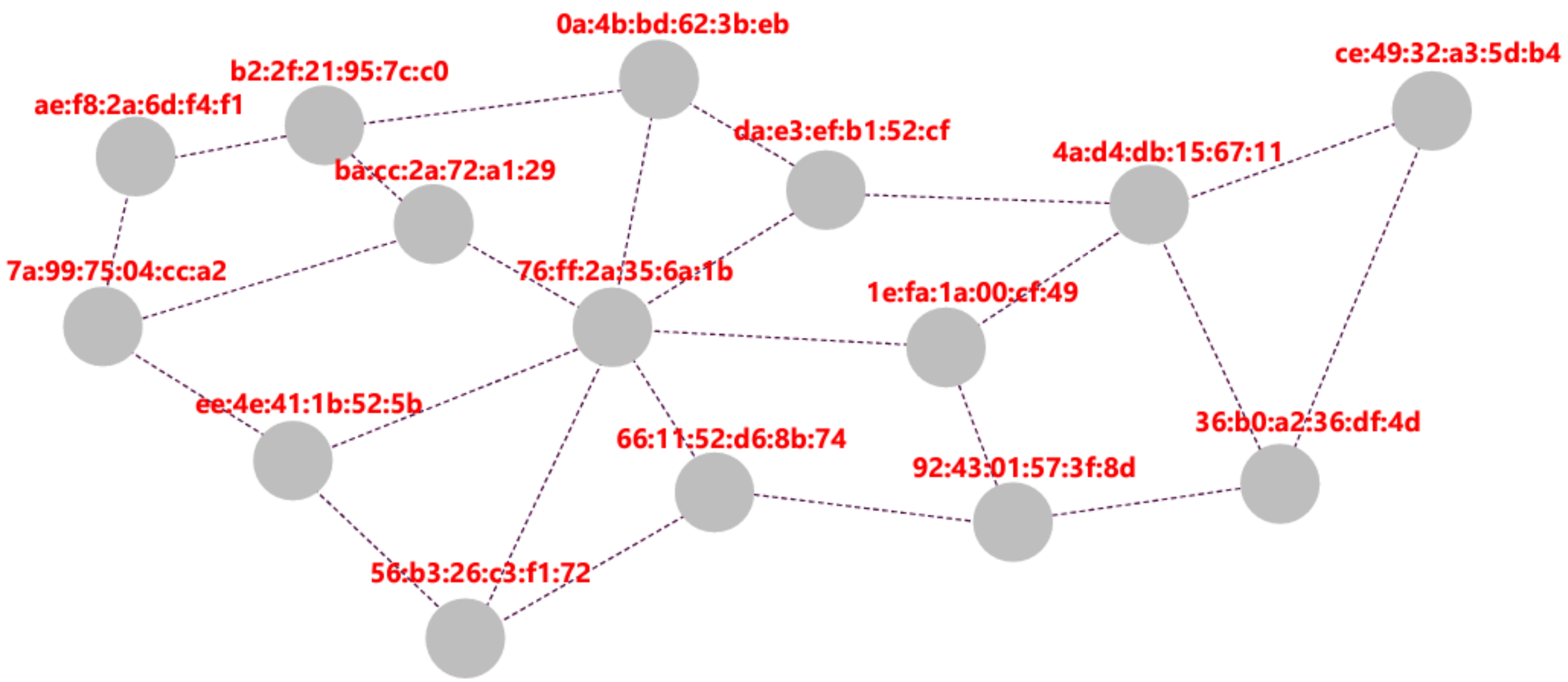
- Mesh networking consists in managing connections between networking elements in a dynamic way to forward data.
  - dynamically self-organise and self-configure.

- To do that, we use distributed algorithms that need to solve **key challenges:**
  - addressing and identifying nodes
  - routing across multiple hops

5.3.1: Addressing

Hardware addresses

- Interface/node comes with built-in address/key from factory (e.g., MAC address)
  - e.g., vendor given block from standard body, allocates to production line

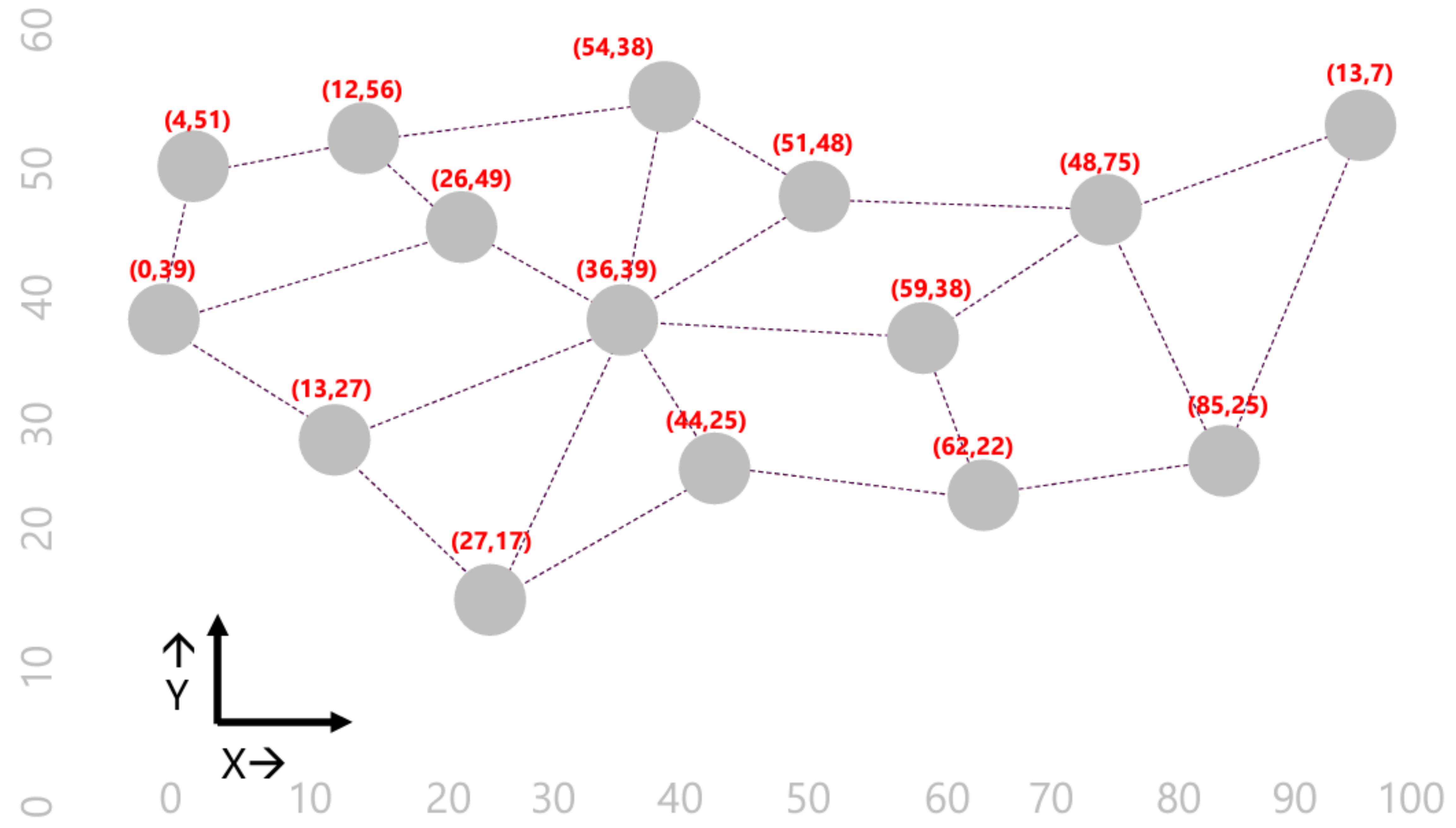


- Benefit: no address assign protocol needed

- Downsides: addresses might be too long, harder to route on addresses

Geographic addressing

- Addresses are assigned depending on the location of the devices (geographical- non topological).
- Define a coordinate system and assign addresses (2D or 3D) to nodes based on that, e.g., using a GPS system.

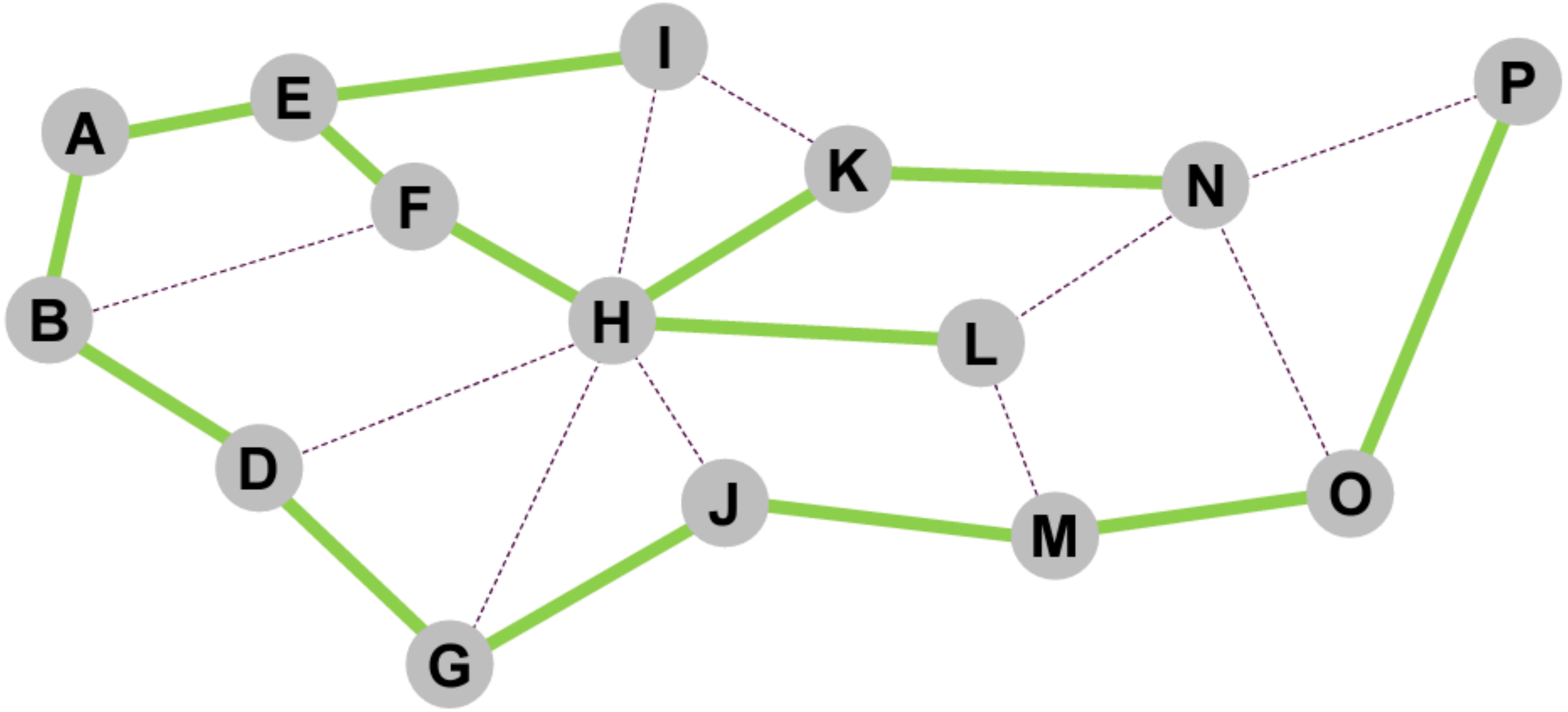


- Can simplify routing but requires mechanism for coordinate assignment (e.g., GPS)
- Benefit: no address assignment protocol needed.
- Downsides: addresses might be too long (small areas, a lot of decimals or address collisions).

Hierarchical Addressing (1)

- Idea: impose a tree on top of our topology and then constrain addresses according to the tree.

- Select a node as the root of the tree (usually, a reliable node, powered and connected through wires).



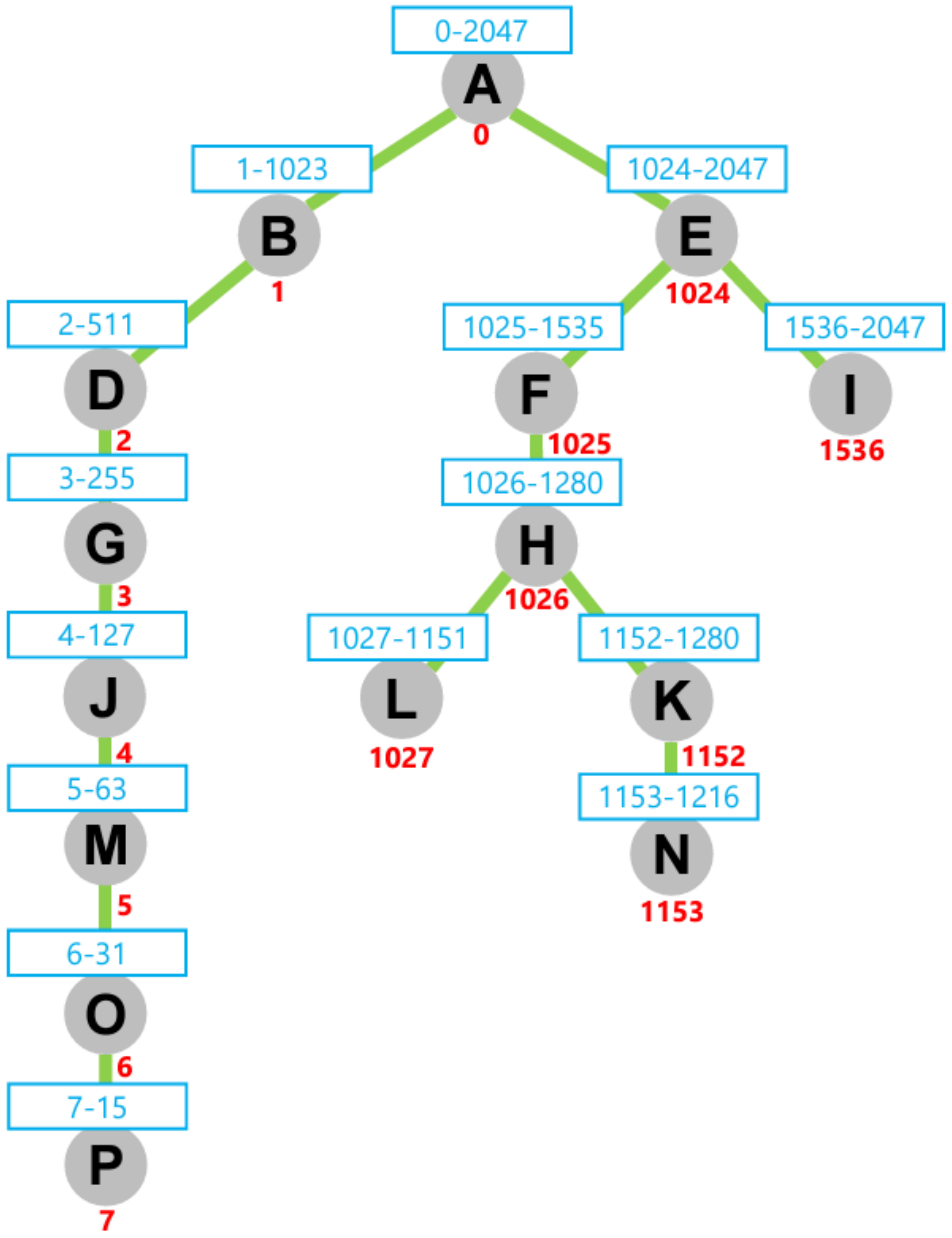
- Assign a set of addresses to the root, which uses one address for itself and assigns the remaining ones to its children

Hierarchical Addressing (2)

- Uses a simple distributed protocol to do so, simply each node allocates a bunch of addresses for their children.

- Benefits: easier to route on addresses

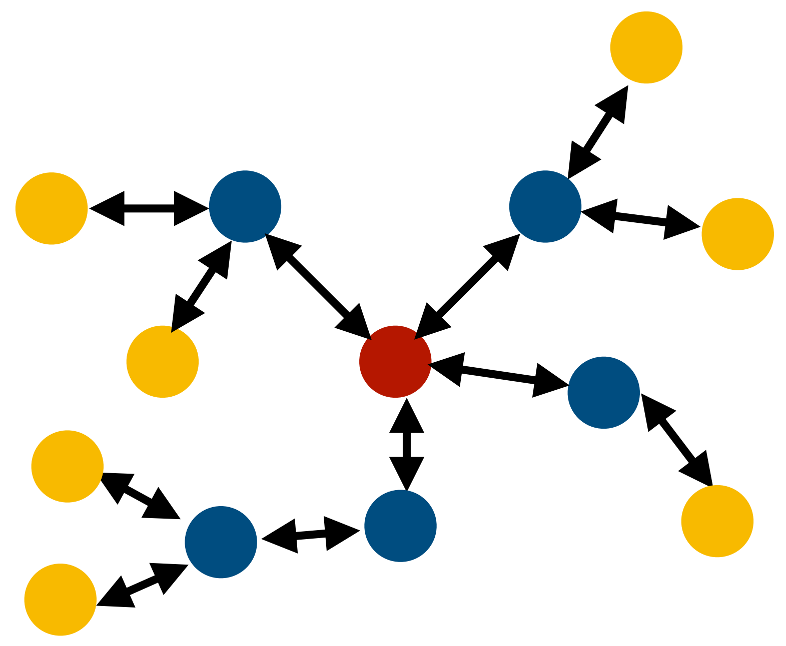
- Downsides: requires tree-like organisation.
  If many new nodes join the network, you do not want to re-allocate addresses. Better to predict a worst case scenario for pre-allocation, but this requires some advance knowledge of future arrival patterns.



Hierarchical addressing: Zigbees's distributed addressing scheme (1)

- Distributed addressing scheme with hierarchical paradigm which assigns each node a unique 16-bit address and makes the following assumptions:
  - tree that it is going to be created has a maximum depth, L,
  - maximum number of children per parent, C,
  - maximum number of forwarding nodes (routers) per parent, R, $(R \leq C)$.

IEEE 802.15.4 tree topology

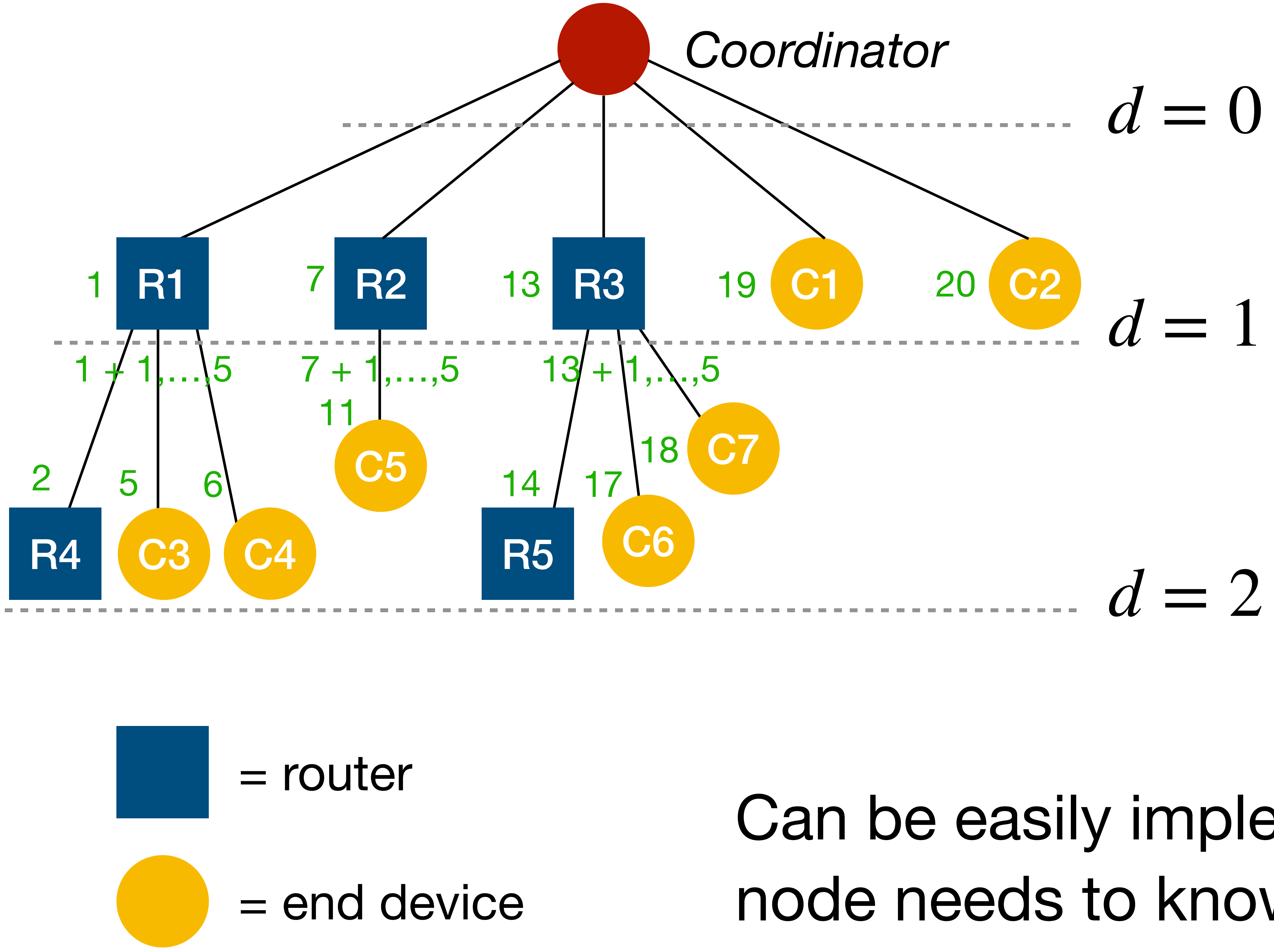


- The address of the n-th router child laying on level $d+1$ is $A_{d+1}^{r,n} = A_{par} + S(d) \cdot (n-1) + 1.$
- The address of the l-th non router child laying on level $d+1$ is $A_{d+1}^{e,l} = A_{par} + S(d) \cdot R + l,\ 1 \leq l \leq C - R.$
- $S(d)$ is the number of nodes in each subtree rooted in a node at level $d+1$.

Hierarchical addressing: Zigbees's distributed addressing scheme (2)

$$S(d) = \begin{cases} 0 & \text{if } R = 0 \\ 1 + C(L - d - 1) & \text{if } R = 1 \\ \frac{CR^{L-d-1} - 1 - C + R}{R - 1} & \text{if } R > 1 \end{cases}$$

Hierarchical addressing: Zigbees's distributed addressing scheme (3)



Example: $L = 2, C = 5, R = 3$

$$S(0) = 6,$$

$$S(1) = 1$$

$$A_1^{r,1} = 0 + S(0) \cdot (1 - 1) + 1 = 1,$$
$$A_1^{r,2} = 0 + S(0) \cdot (2 - 1) + 1 = 7,$$
$$A_1^{r,3} = 0 + S(0) \cdot (3 - 1) + 1 = 13,$$
$$A_2^{r,1} = 1 + S(1) \cdot (1 - 1) + 1 = 2,$$
$$A_2^{e,1} = 1 + S(1) \cdot 3 + 1 = 5,$$

Can be easily implemented in a distributed manner: each node needs to know d, n, L, C, R.

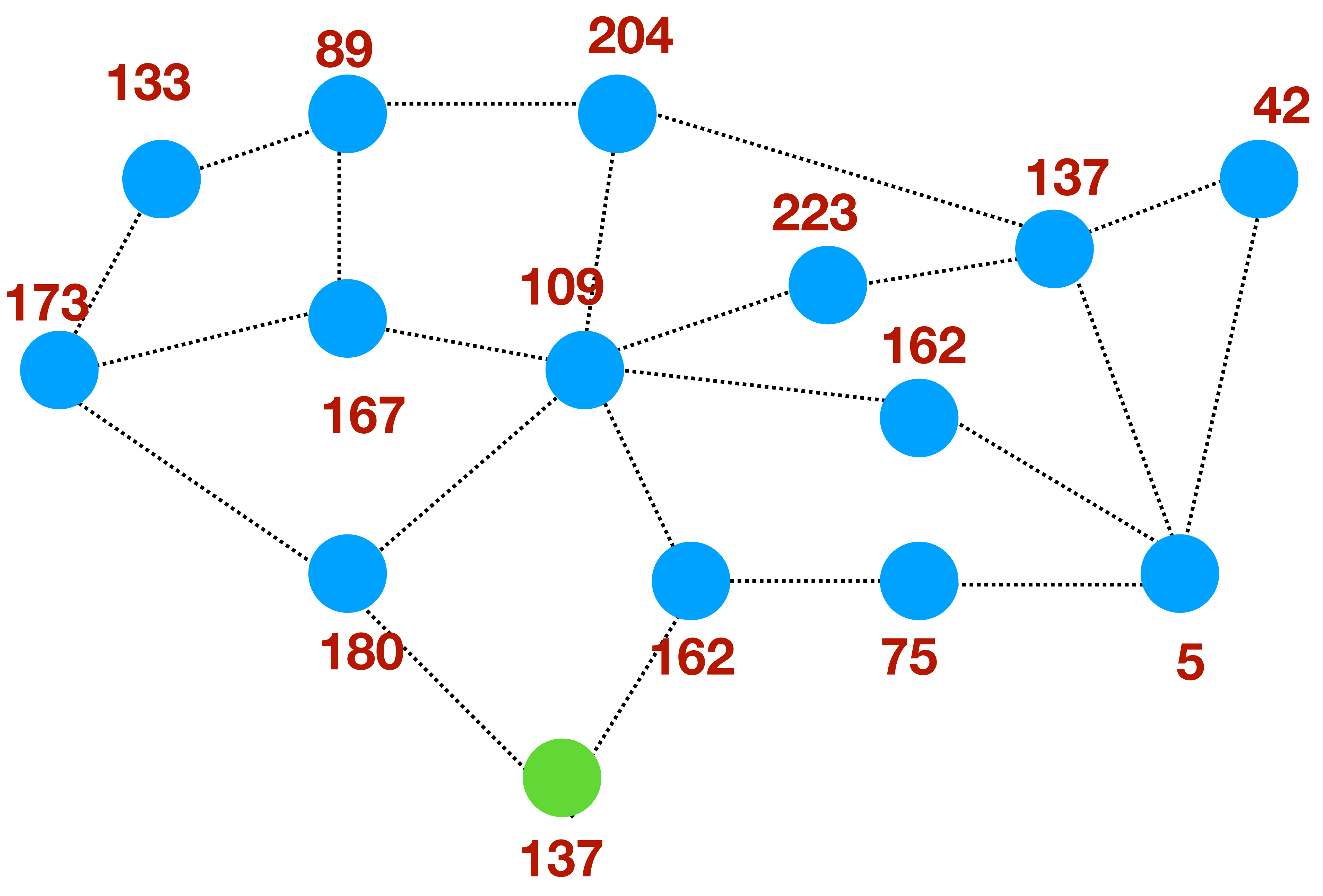
Hierarchical addressing: Zigbees's distributed addressing scheme (2)

$$S(d) = \begin{cases} 0 & \text{if } R = 0 \\ 1 + C(L - d - 1) & \text{if } R = 1 \\ \boxed{\dfrac{CR^{L-d-1} - 1 - C + R}{R - 1}} & \text{if } R > 1 \end{cases}$$

Exercise: motivate this.
hint:
1. count how many routers and end points are in each subtree.
2. Use $\displaystyle\sum_{k=m}^{n} x^k = \frac{x^{n+1} - x^m}{x - 1}$
3. Simplify your expression to obtain the one in the $\boxed{\text{box}}$

Stochastic addressing

- Nodes choose random number as their address
- Simple to implement, but requires conflict resolution to ensure uniqueness - new node joining the network broadcasts its random address and if it is matching the address of another node, the node sends an error message and the node chooses another address
- More common than one might expect (see birthday problem)
- Addresses are location-independent and trickier to route.



5.3.2 Mesh Routing

- Key approaches:
  1) “We should always have routes available to everyone, at all times” - **proactive routing.**
     - better for fixed/static environments, frequent communication.
     - lower route acquisition time.
     - Requires more control overhead, memory, power.

  2) “We should create routes only when we need them” - **reactive routing.**
     - Better for dynamic environments, rare communication.
     - higher route acquisition time.
     - Requires less control overhead, memory and power.

- Before talking about Mesh Routing, let’s revise traditional routing approaches: Link State Routing and Distance Vector Routing.

Link State Routing (1)

- Each node maintains a map of the topology of the network called “topology database”.

- Nodes exchange information about their knowledge of the network.

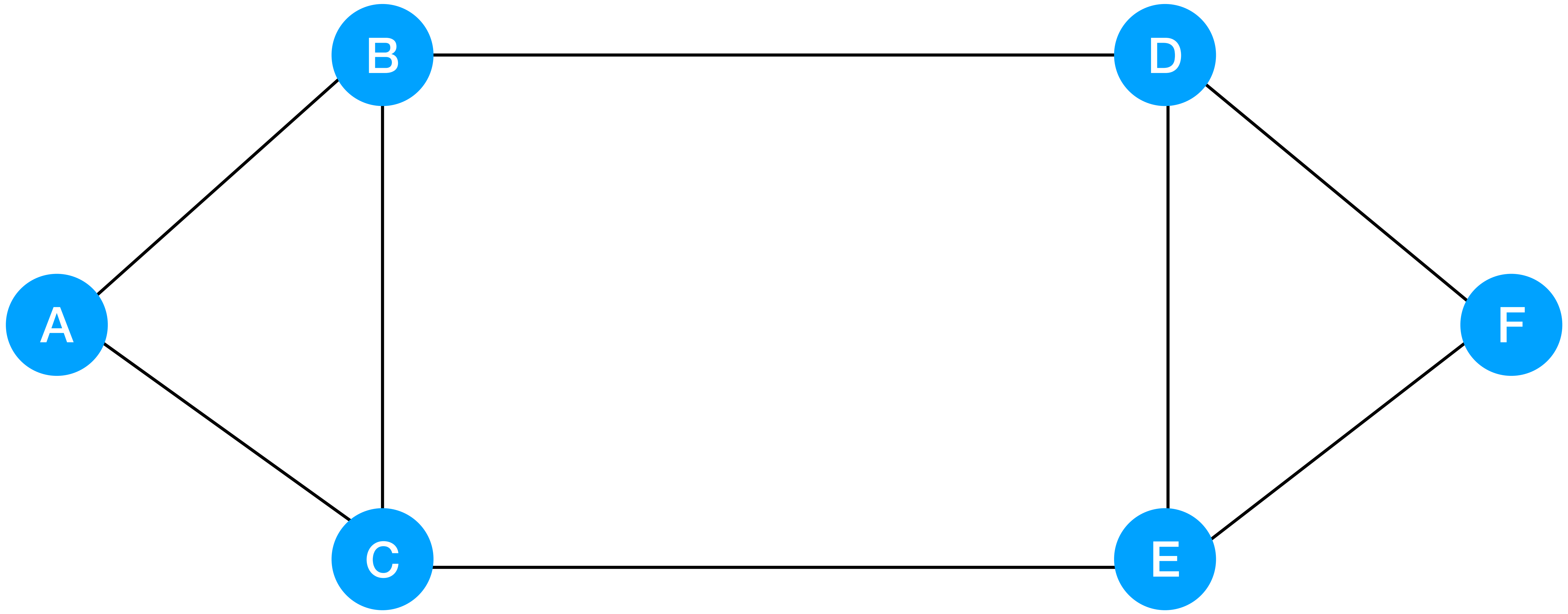
- Each node looks at its local topology (initially, nodes discover their neighbours via hello-based protocols) and floods a series of link-state advertisement.

- This operation is repeated as a new nodes join the network and as nodes fail.

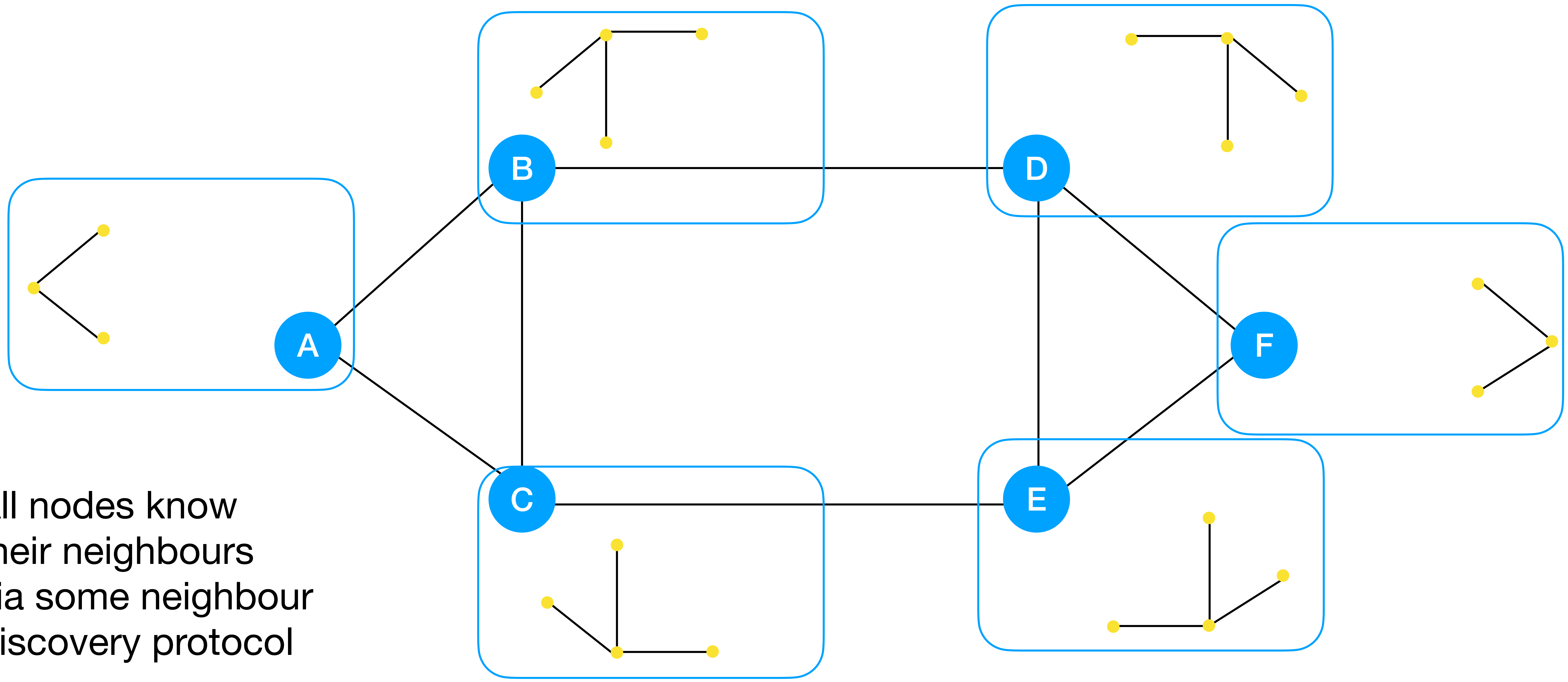
- Once a node knows the topology of the network, it runs Dijkstra to find the next hop of the shortest path to reach the destination.

- Links can be weighted

Link State Routing (2)

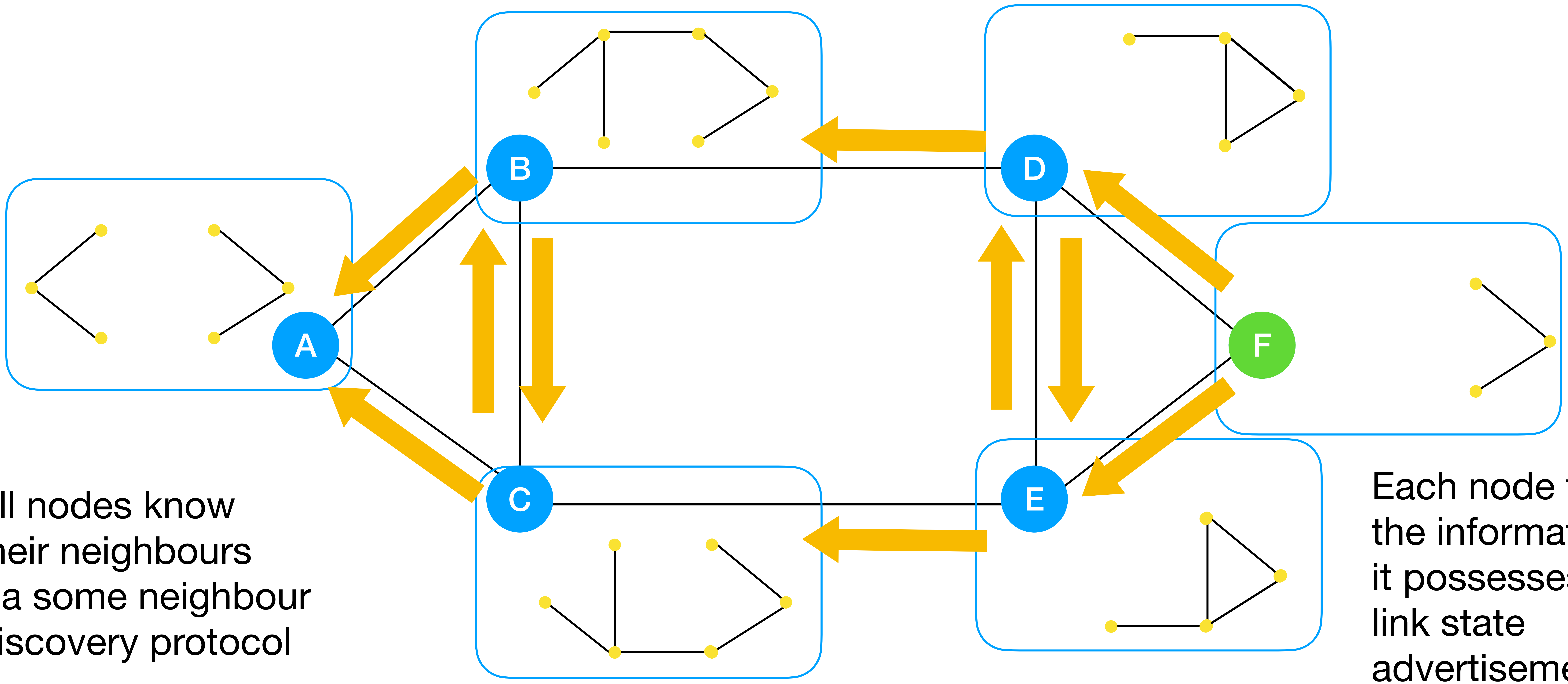


Link State Routing (2)



All nodes know
their neighbours
via some neighbour
discovery protocol

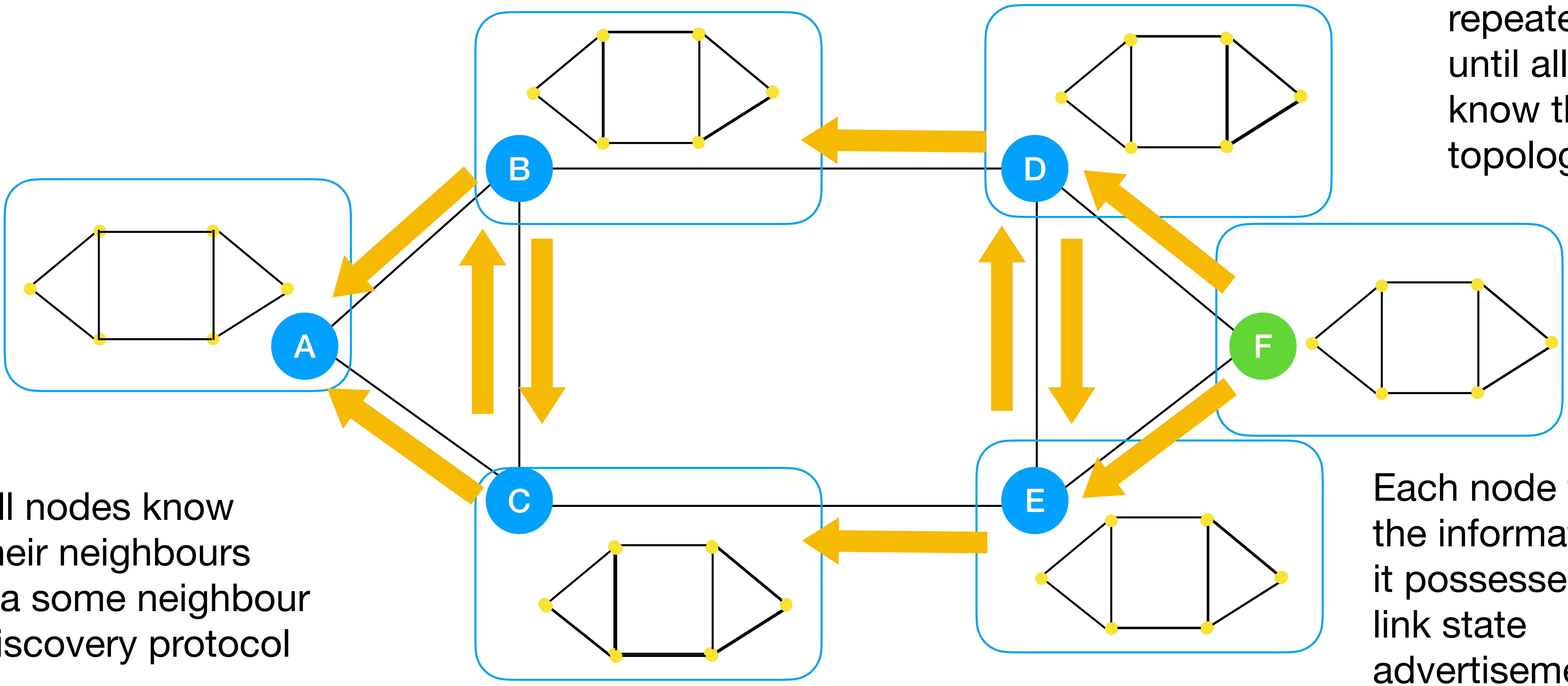
Link State Routing (2)



All nodes know
their neighbours
via some neighbour
discovery protocol

Each node floods
the information
it possesses through
link state
advertisements.
Nodes update their
map of the network

Link State Routing (2)



The process is repeated until all nodes know the entire topology

All nodes know their neighbours via some neighbour discovery protocol

Each node floods the information it possesses through link state advertisements. Nodes update their map of the network

Link State Routing (3)

- How do we prevent this mechanism to stop?
  - Packets come with a sequence number, identifying the event that caused the transmission of a link-state update. If a router sees that it has already transmitted a packet with a certain sequence number, it stops.

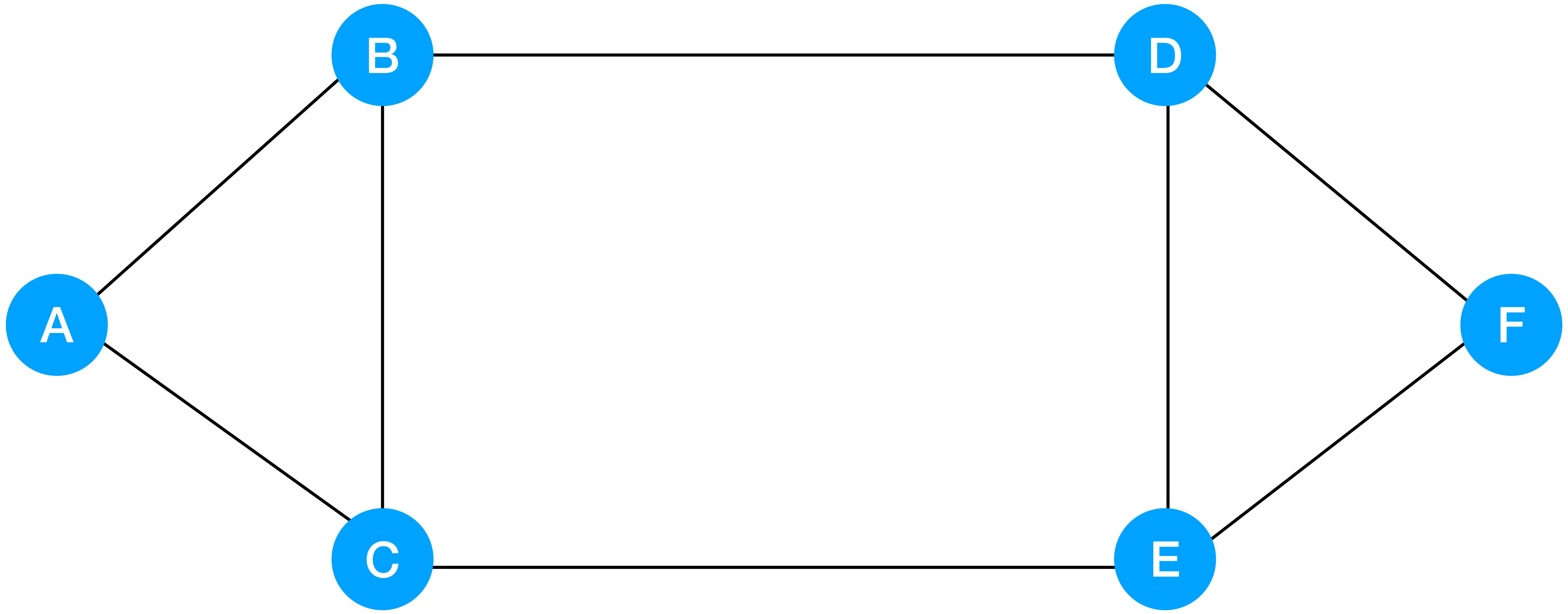
- What happens if a new node appears but flooding already happened?
  - The new node asks its neighbours a dump of their topology database and advertises its existence to the network

- Once each node knows the network, it computes the shortest path to the destination
  - done with Dijkstra, but only the first time: nodes store in the routing table the next hop in the shortest path to a destination node.
  - protocols can assign weights to links

Distance Vector Routing (1)

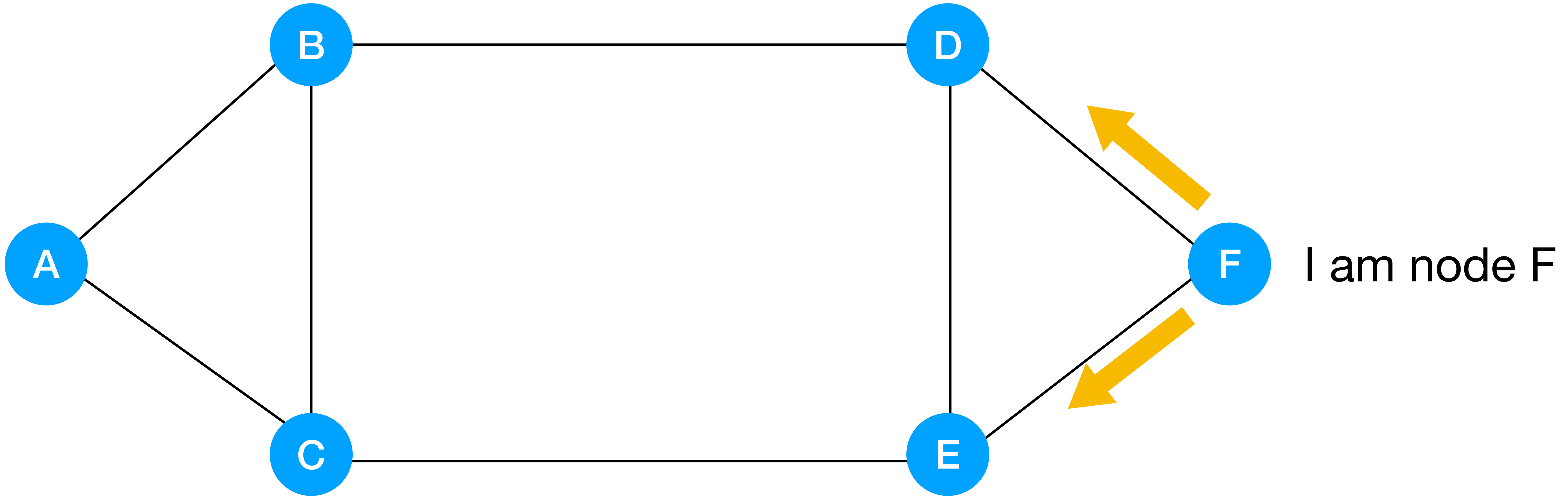
- Each router knows the links to its neighbours (does not flood this information to the whole network)
- Each router has provisional “shortest path” to every other router (e.g., node A knows the cost of getting to router B)
- Nodes exchange this distance vector information with their neighbouring routers
- Routers look over the set of options offered by their neighbours and select the best one (that is, the one with smallest cost/weight).
- Iterative process, converges to set of shortest paths

Distance Vector Routing (2)



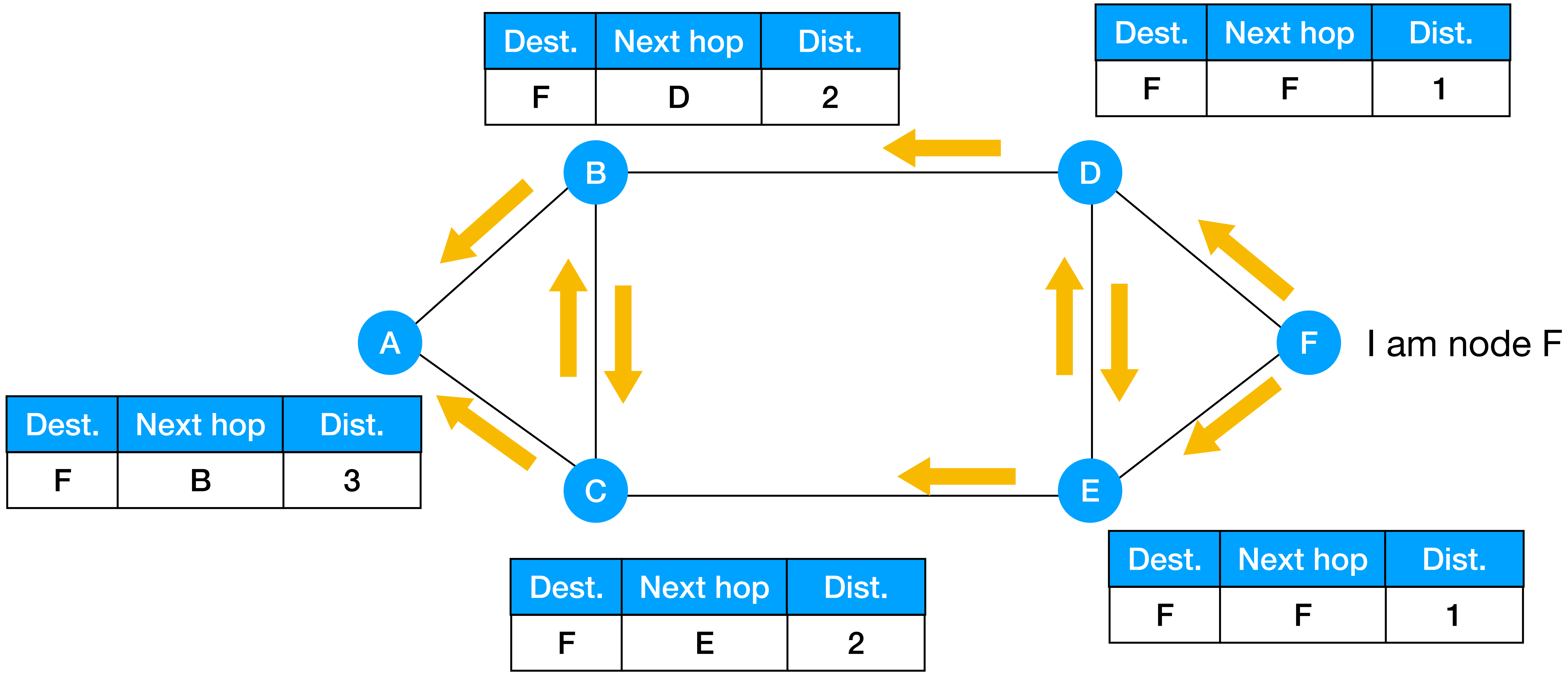
Distance Vector Routing (2)

Dest.	Next hop	Dist.
F	F	1



Dest.	Next hop	Dist.
F	F	1

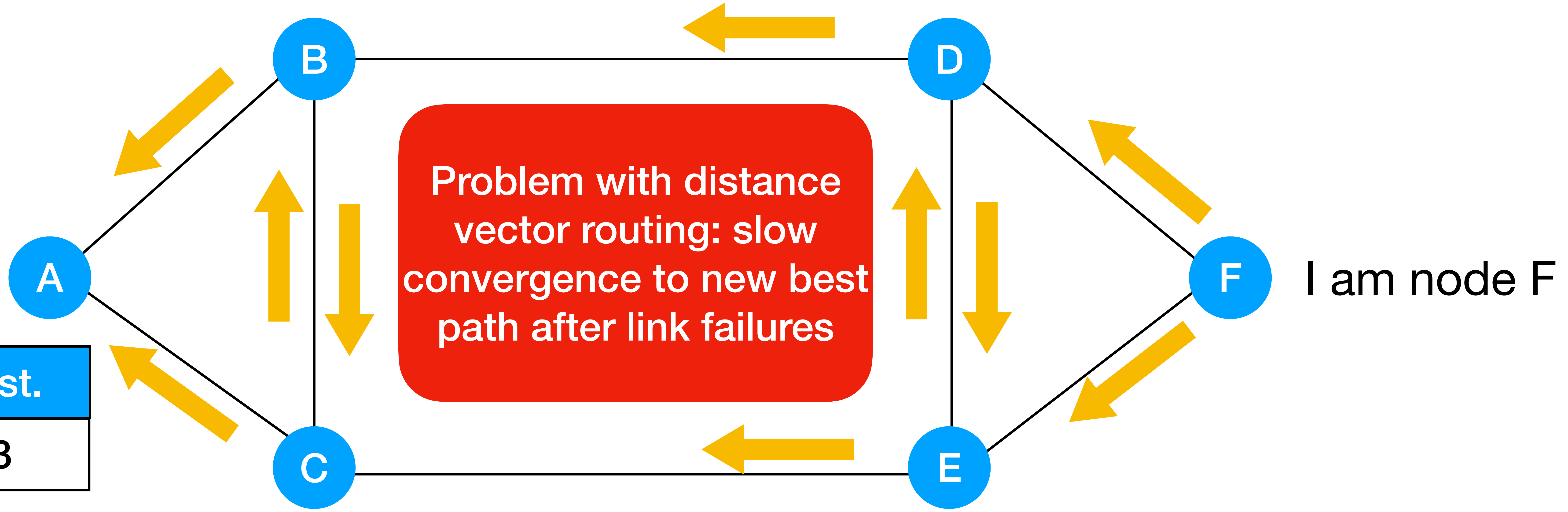
Distance Vector Routing (2)



Distance Vector Routing (2)

Dest.	Next hop	Dist.
F	D	2

Dest.	Next hop	Dist.
F	F	1



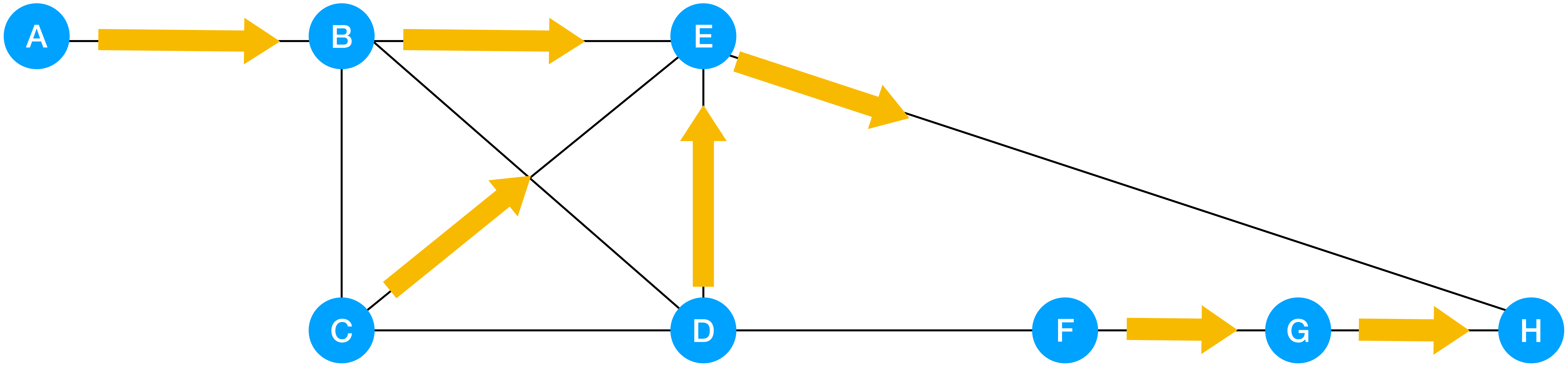
Dest.	Next hop	Dist.
F	B	3

Dest.	Next hop	Dist.
F	E	2

Dest.	Next hop	Dist.
F	F	1

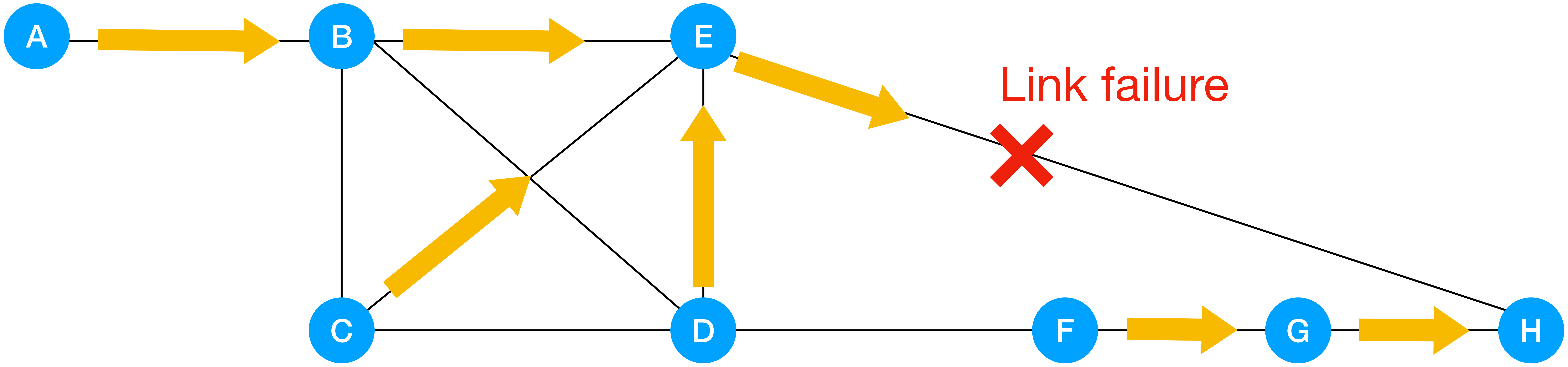
Distance Vector routing - convergence

Next hop from any node to H in shortest path



Distance Vector routing - convergence

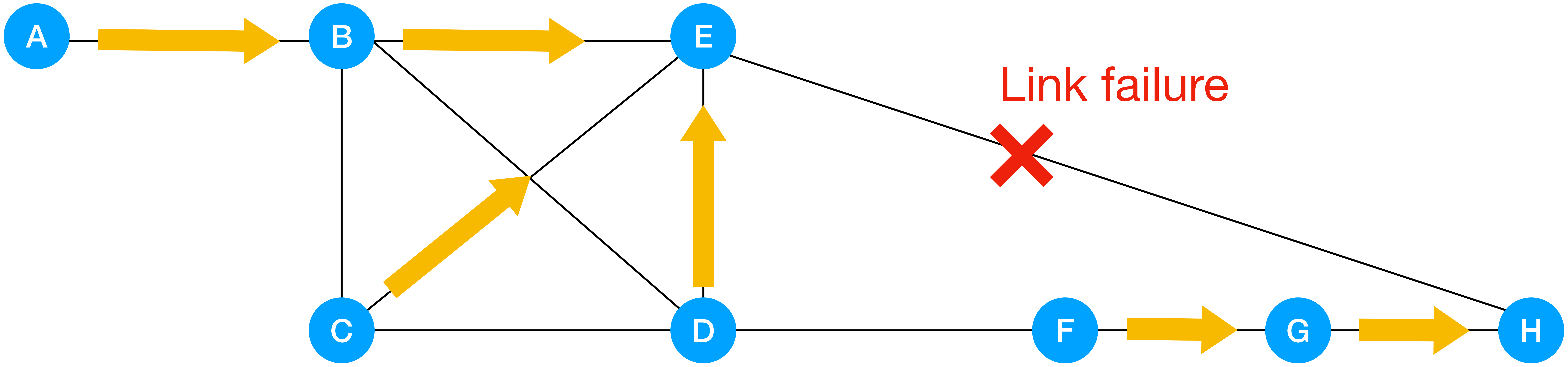
Next hop from any node to H in shortest path



Distance Vector routing - convergence

Next hop from any node to H in shortest path

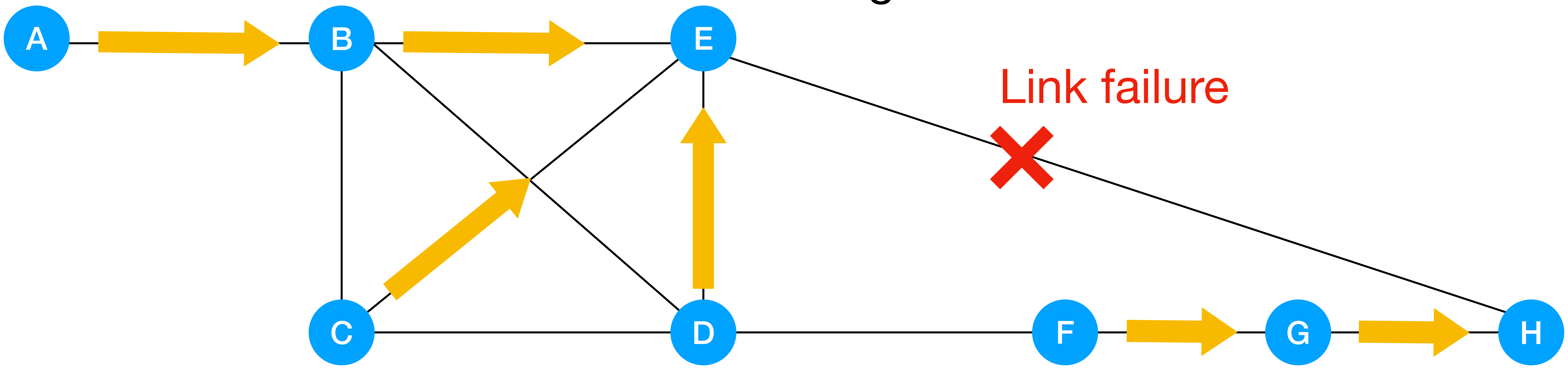
E notices (tries to talk to H but fails)



Distance Vector routing - convergence

Next hop from any node to H in shortest path

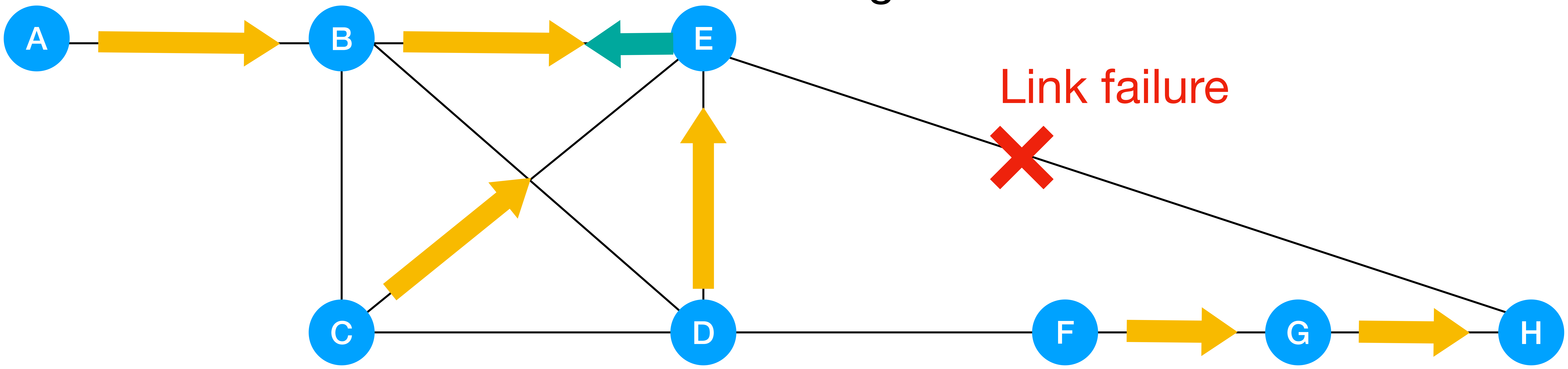
E notices (tries to talk to H but fails)
E sends out withdraw messages
to its neighbours



Distance Vector routing - convergence

Next hop from any node to H in shortest path

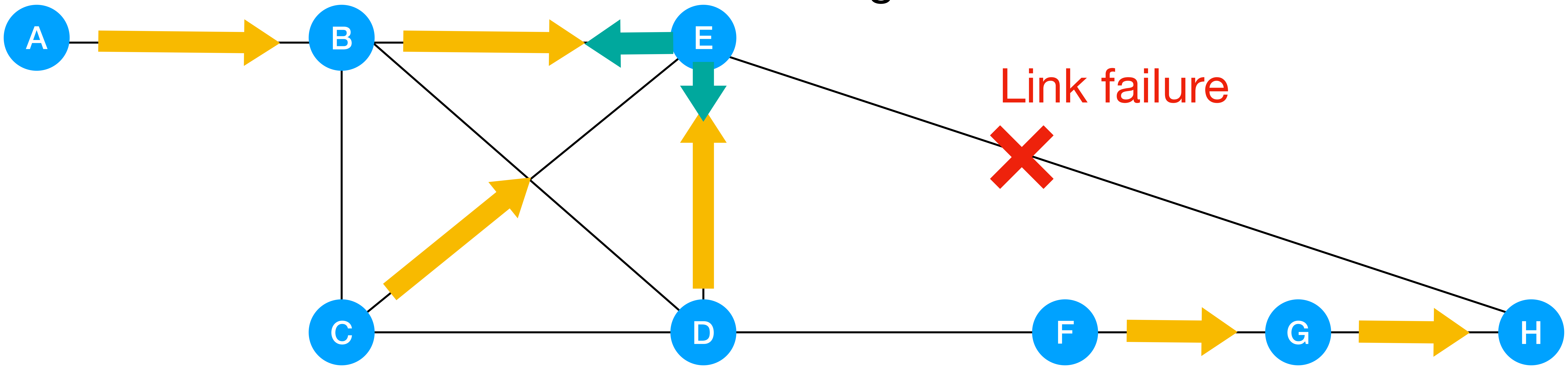
E notices (tries to talk to H but fails)
E sends out withdraw messages
to its neighbours



Distance Vector routing - convergence

Next hop from any node to H in shortest path

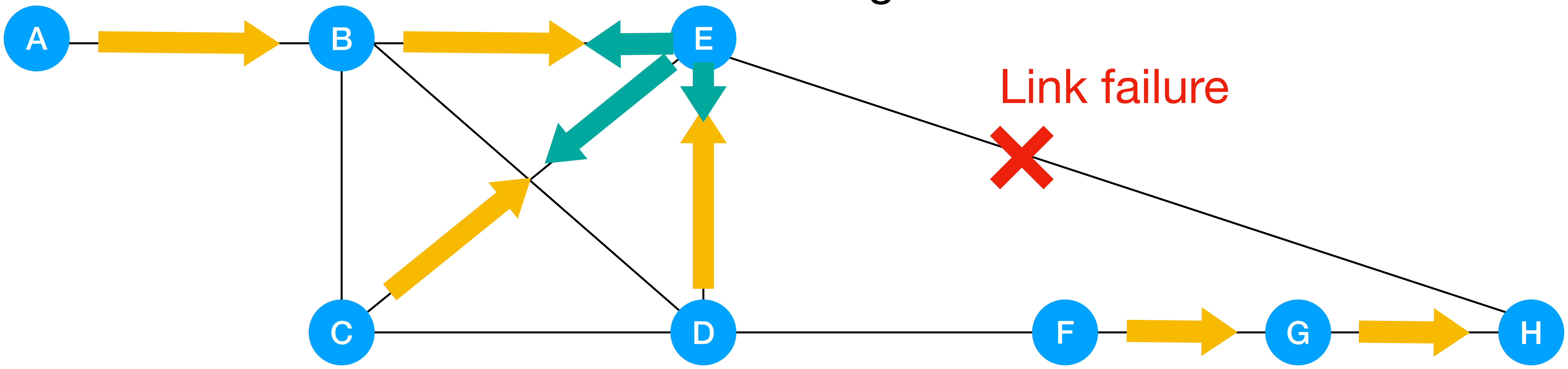
E notices (tries to talk to H but fails)
E sends out withdraw messages
to its neighbours



Distance Vector routing - convergence

Next hop from any node to H in shortest path

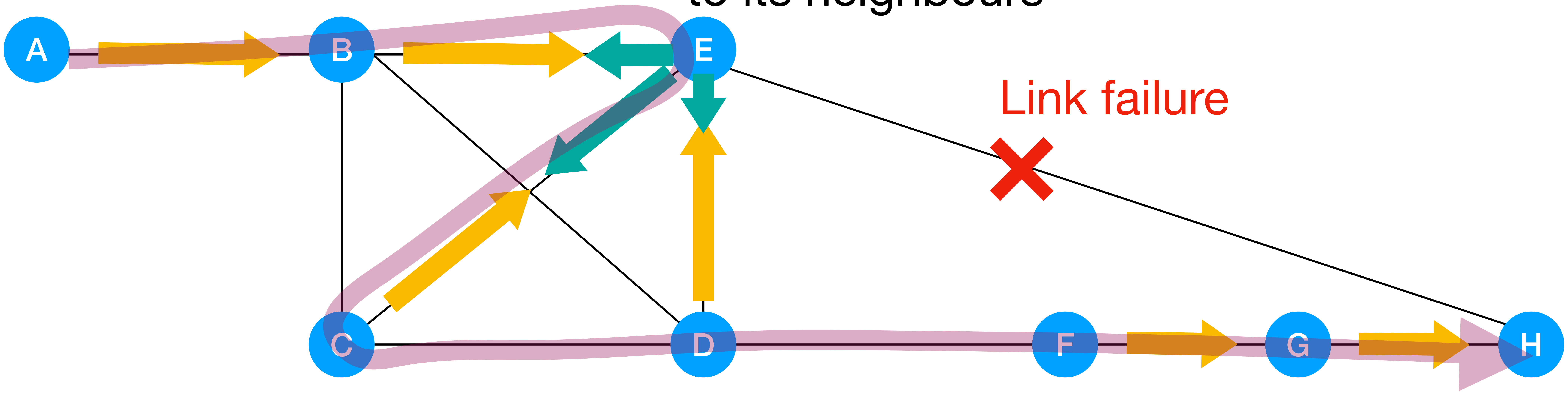
E notices (tries to talk to H but fails)
E sends out withdraw messages
to its neighbours



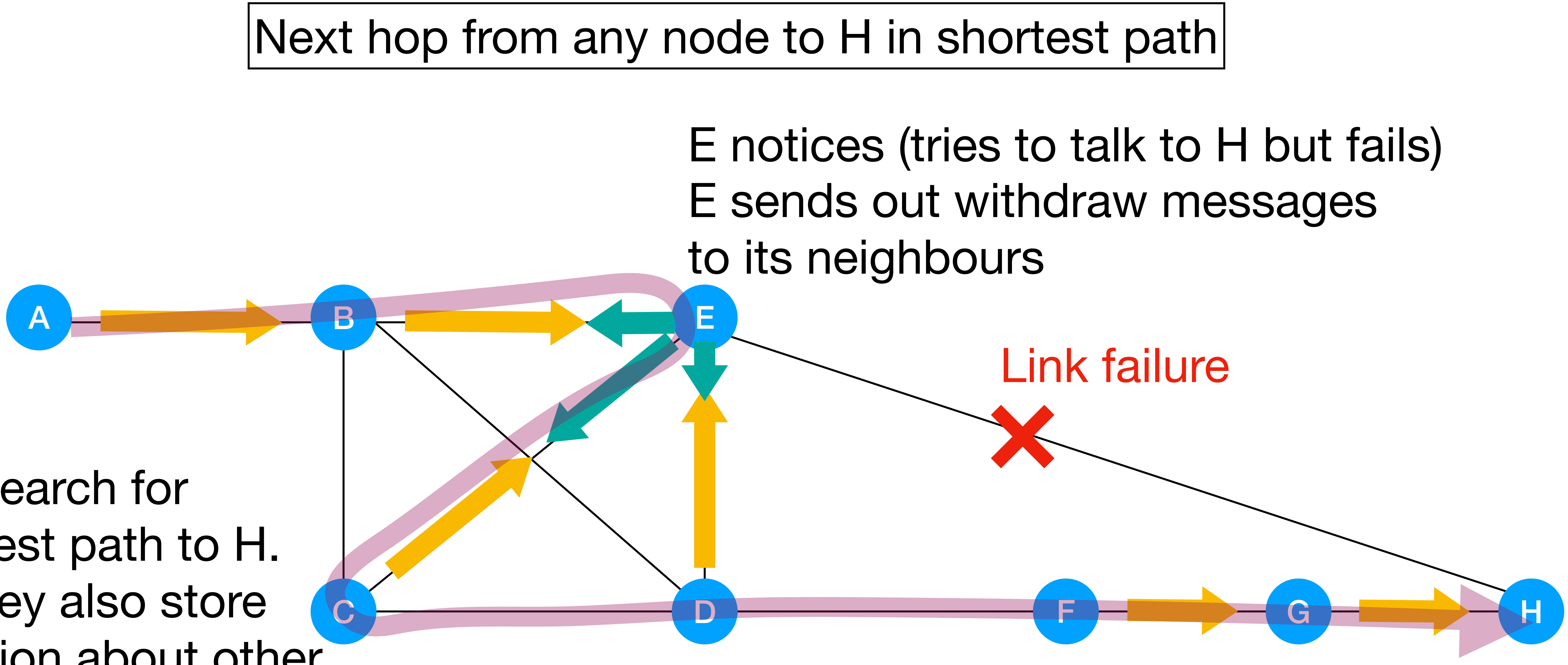
Distance Vector routing - convergence

Next hop from any node to H in shortest path

E notices (tries to talk to H but fails)
E sends out withdraw messages
to its neighbours



Distance Vector routing - convergence



Nodes search for a new best path to H. Since they also store information about other nodes sponsoring their paths, it might take a while before they converge a new best path.

Mesh Routing

- Link state and distance vector are not optimised for battery and bandwidth:
  - in wireless communication, you get local broadcast “for free” (signal propagates out in all directions). No need do unicast all the neighbours.
  - not always need all routes to all nodes.

- In Mesh Networking, used in wireless environments, these factors need to be accounted for.

  - Node cooperate to efficiently route data

  - Nodes can act as relays, enabling multi-hop forwarding

  - Mesh networks dynamically self-organise and self-configure
    - reduce management and configuration overhead
    - improves fault-tolerance
    - dynamically distributes workloads

Need efficient algorithms to achieve this

Mesh Networking Algorithms

1. Optimised Link State Routing (OLSR)
2. Dynamic Source Routing (DSR)
3. Ad Hoc On-demand Distance Vector (AODV)
4. Tree routing
5. Geographic routing
6. Gossip algorithm

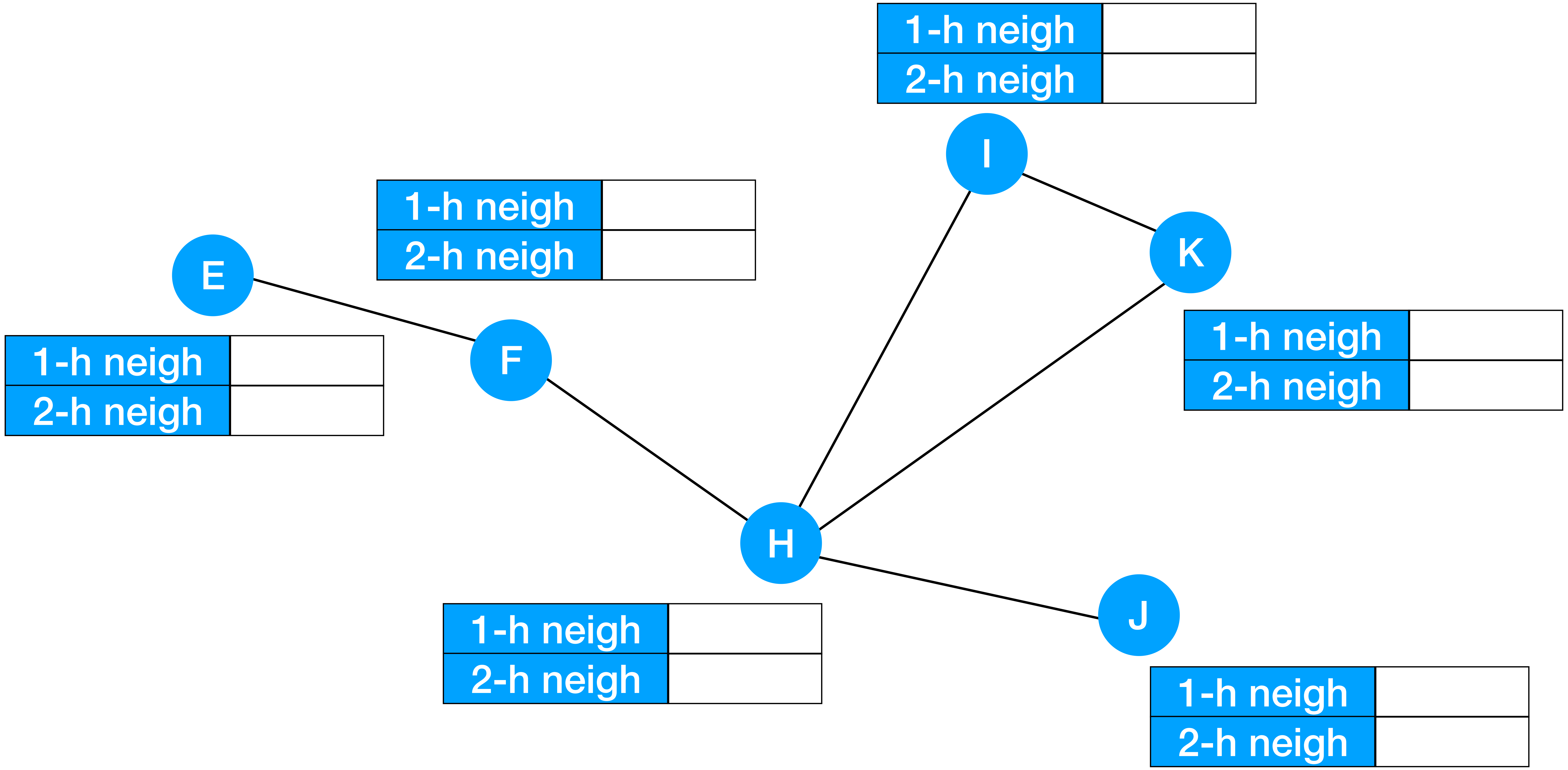
1 Optimised Link State Routing (OLSR) (1)

- Distributed, **proactive** routing protocol that operates as link state, but avoids unnecessary updates.

- Nodes send broadcast messages to reach all neighbours.

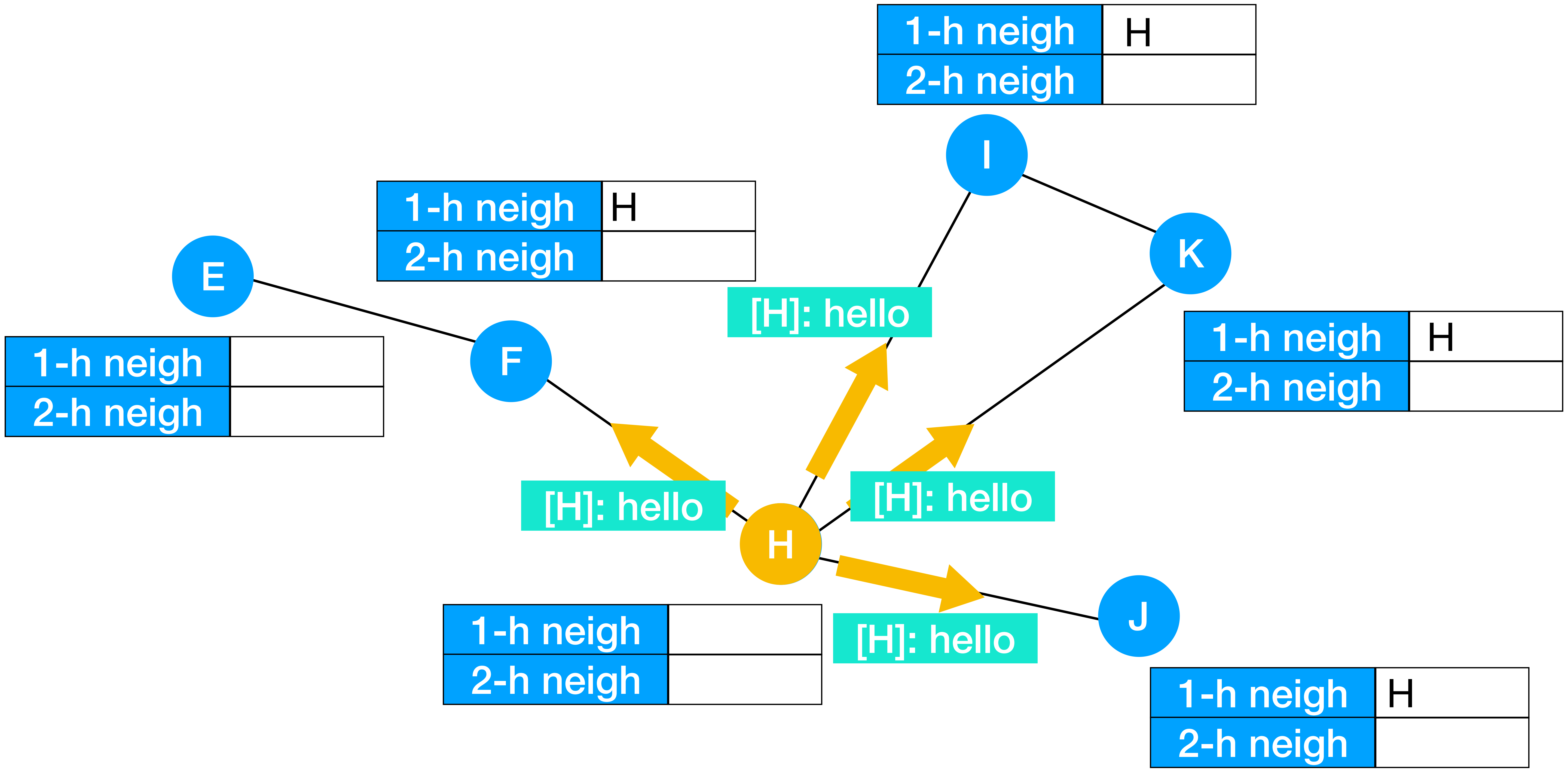
- Each node decides which of its neighbours will forward packets
  - These neighbours are known as **multipoint relays (MPRs)**
  - MPRs selected based on nodes neighbours can reach (want to cover all two-hop neighbours)
  - Only MPRs transmit updates, other nodes do not

1 Optimised Link State Routing (OLSR) (2)



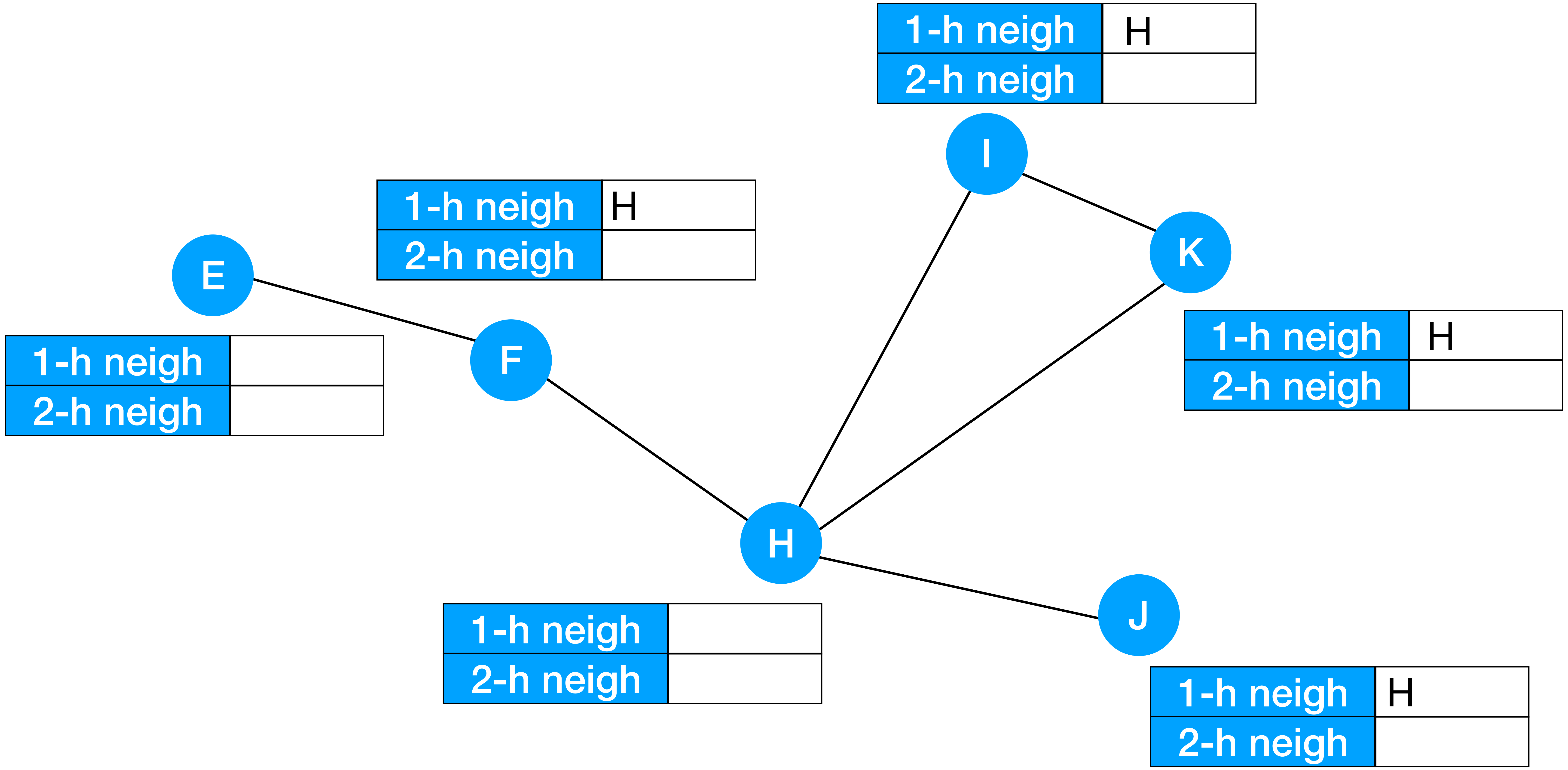
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



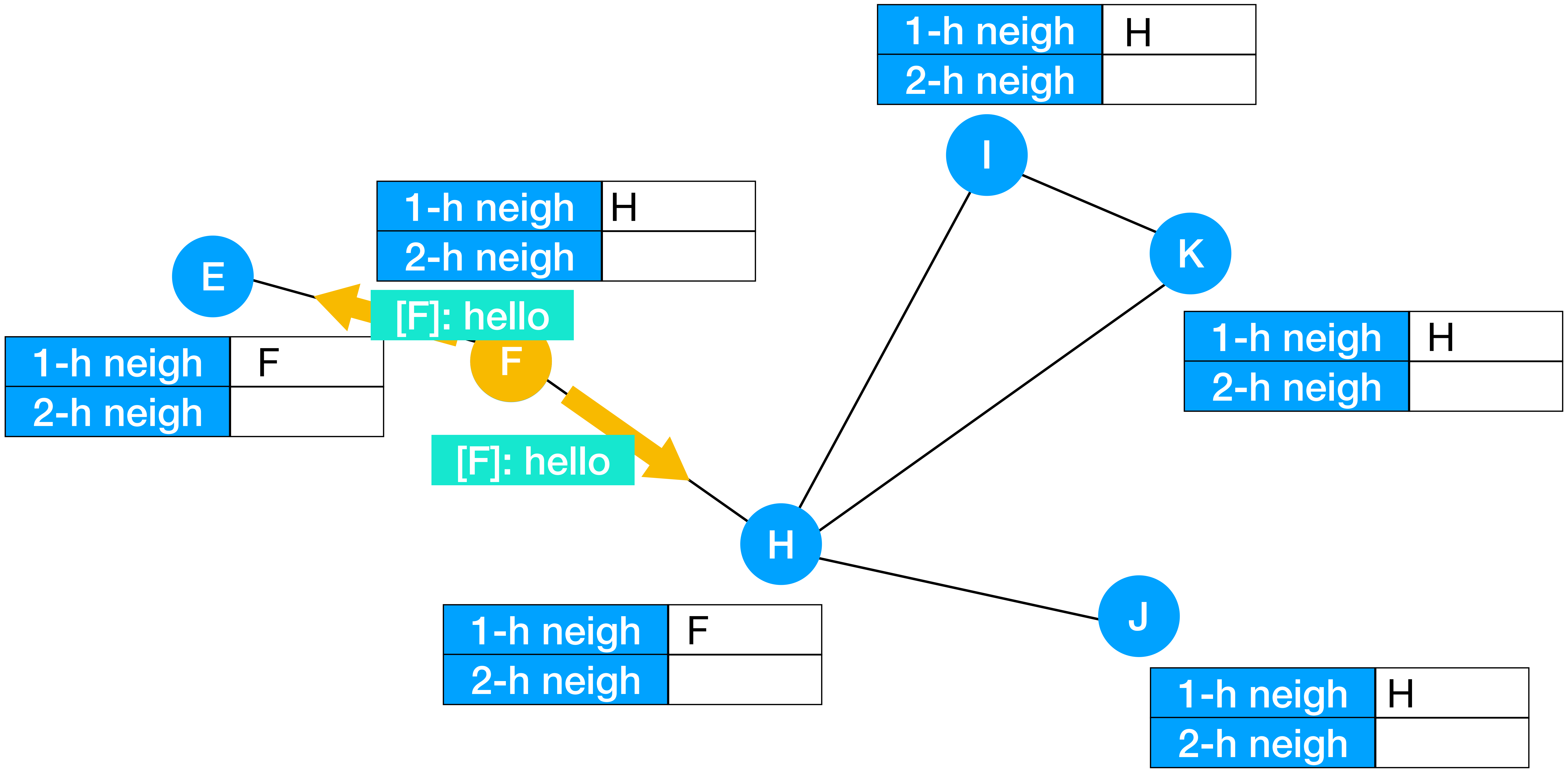
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



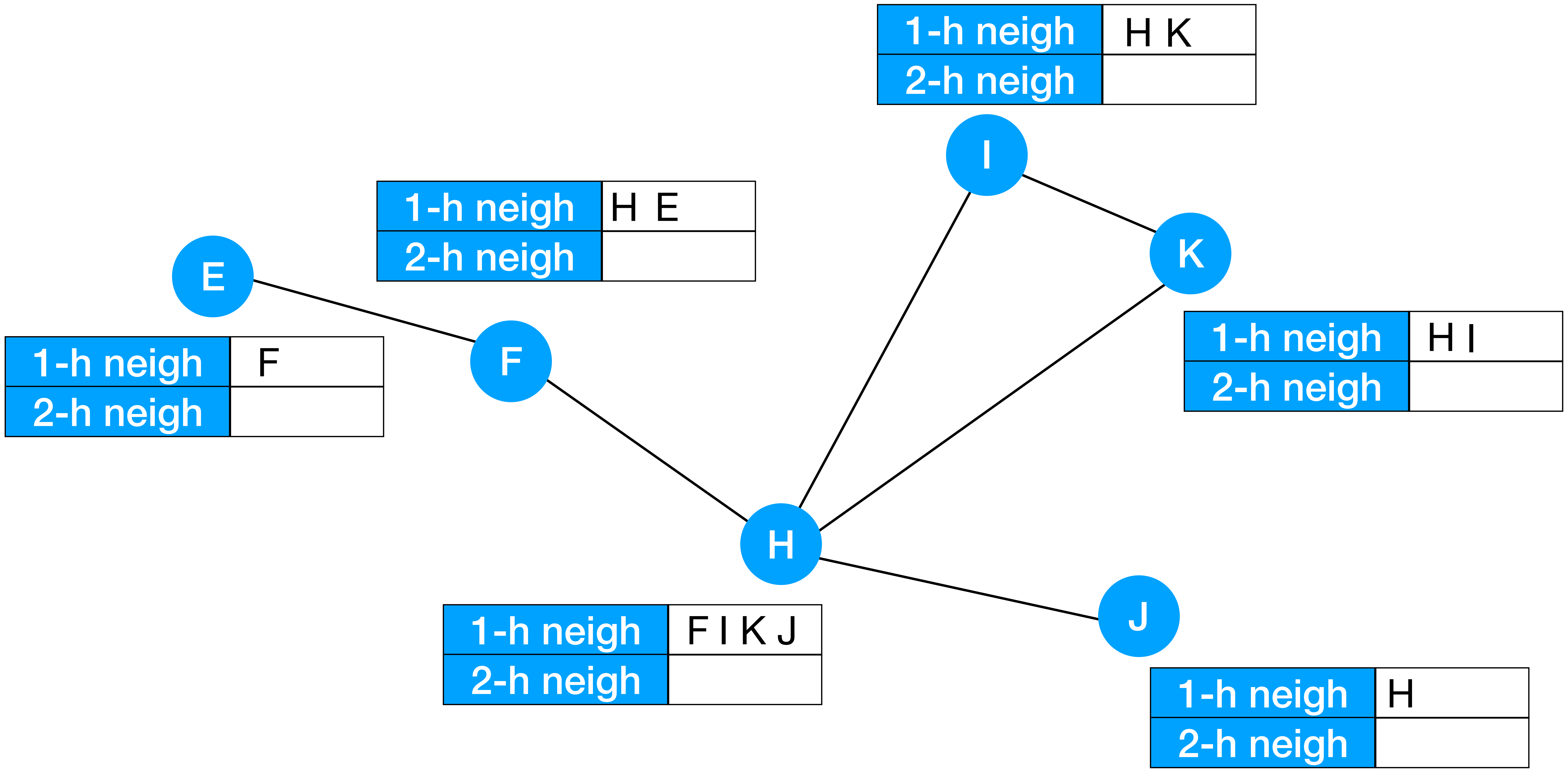
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



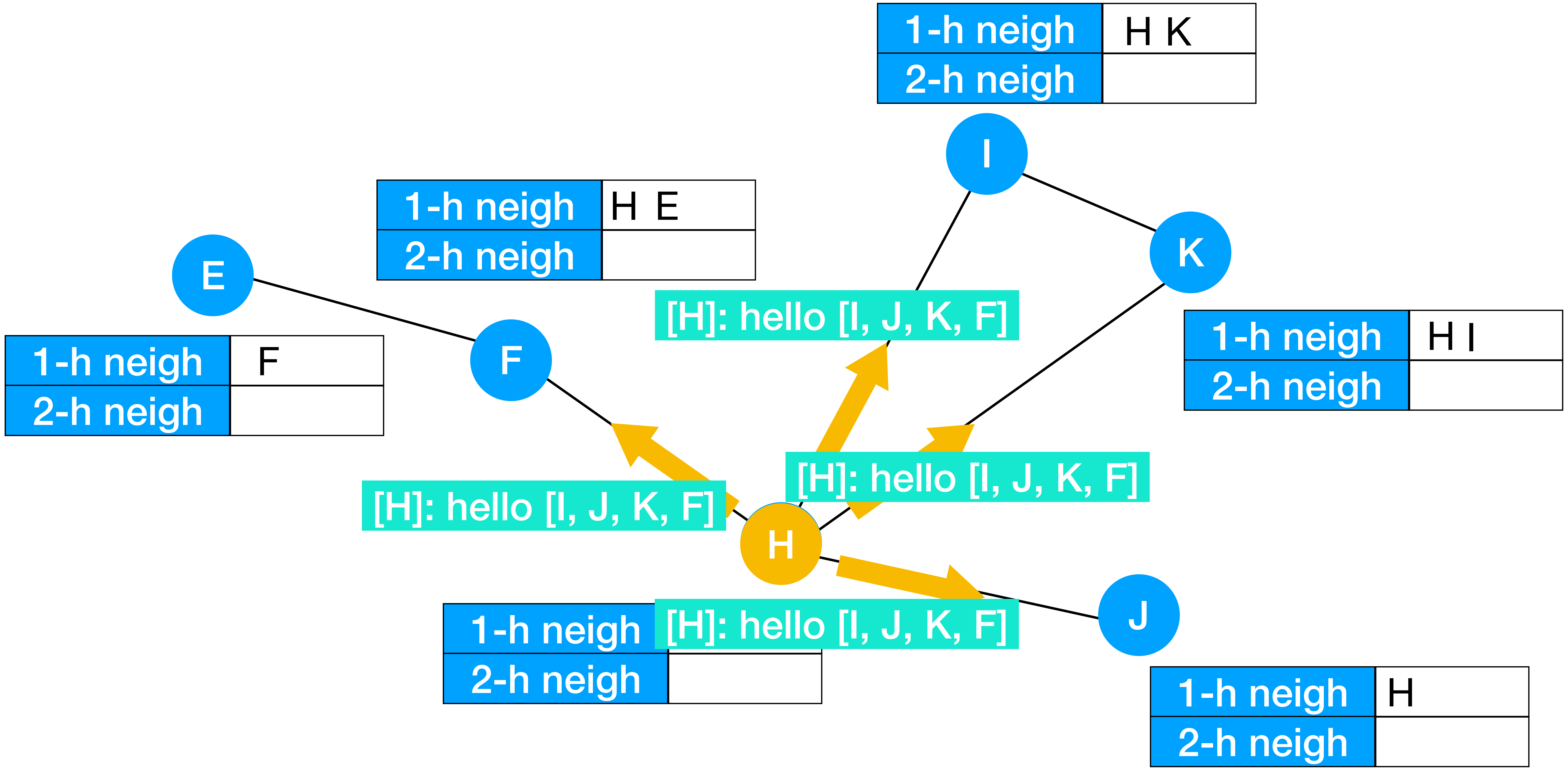
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



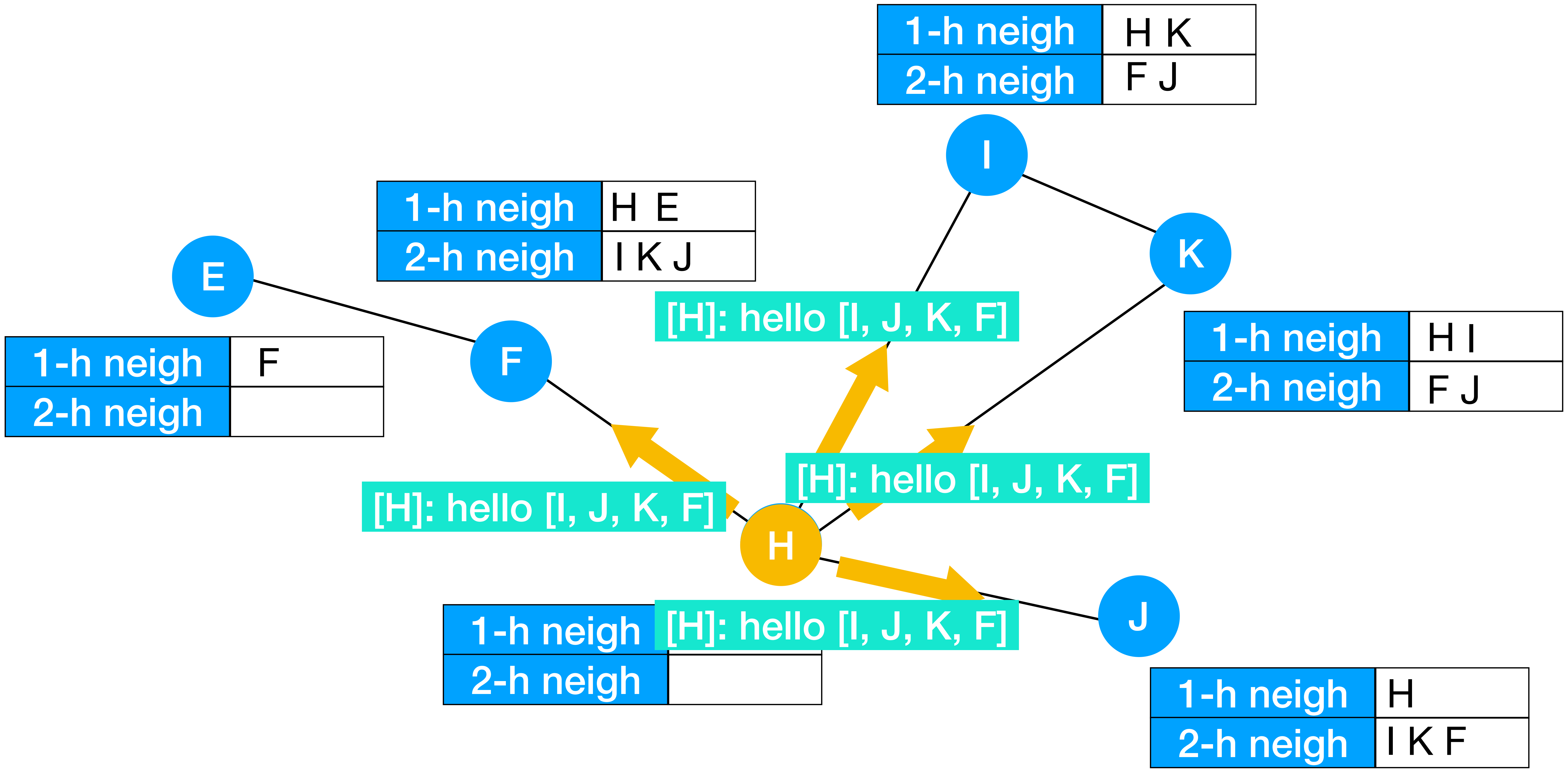
- Step 1: Each node discovers its two-hop neighbours proactively by sending "hello" packets with a list of the sender's neighbour set

1 Optimised Link State Routing (OLSR) (2)



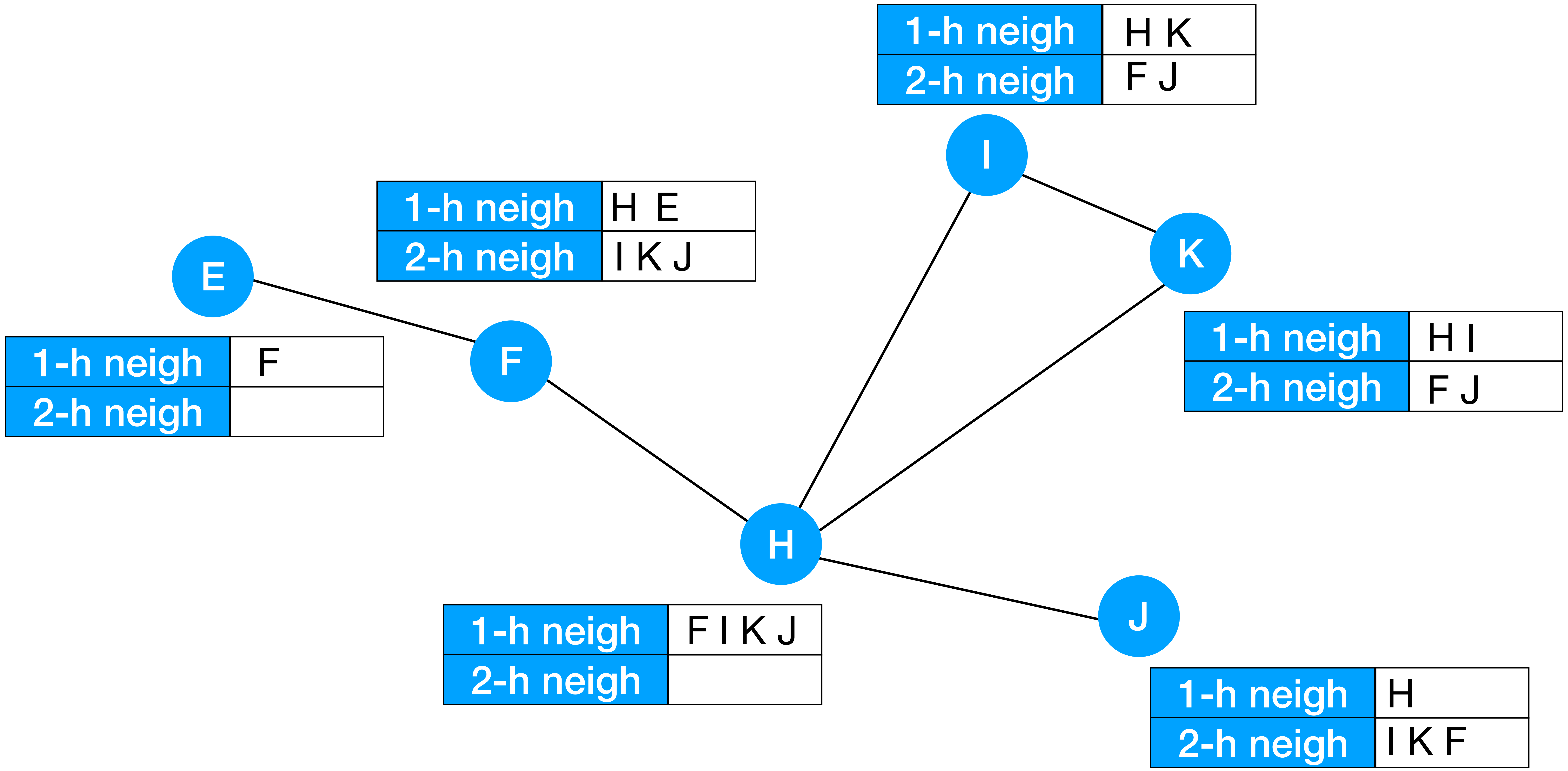
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



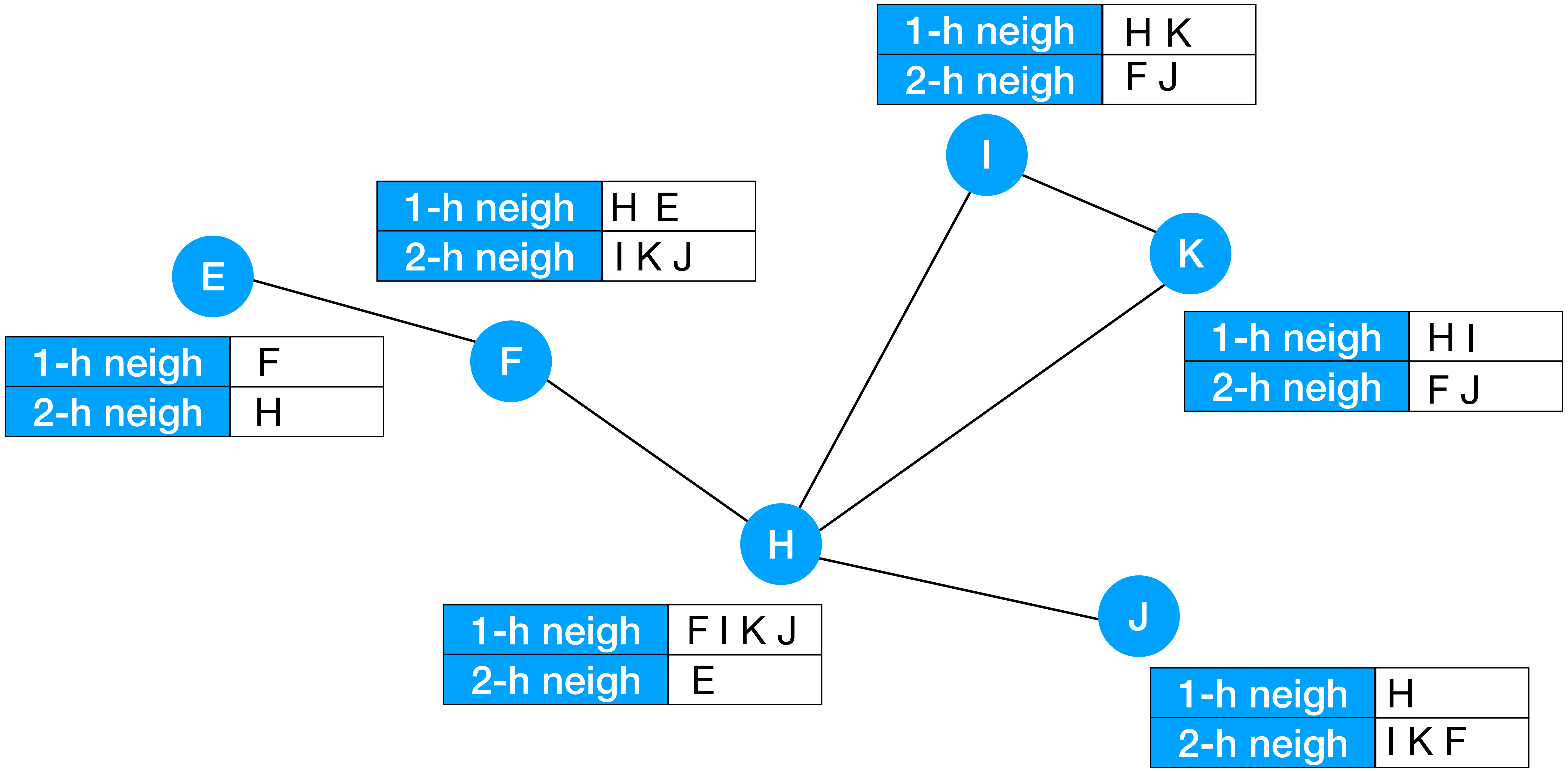
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

Optimised Link State Routing (OLSR) (2)



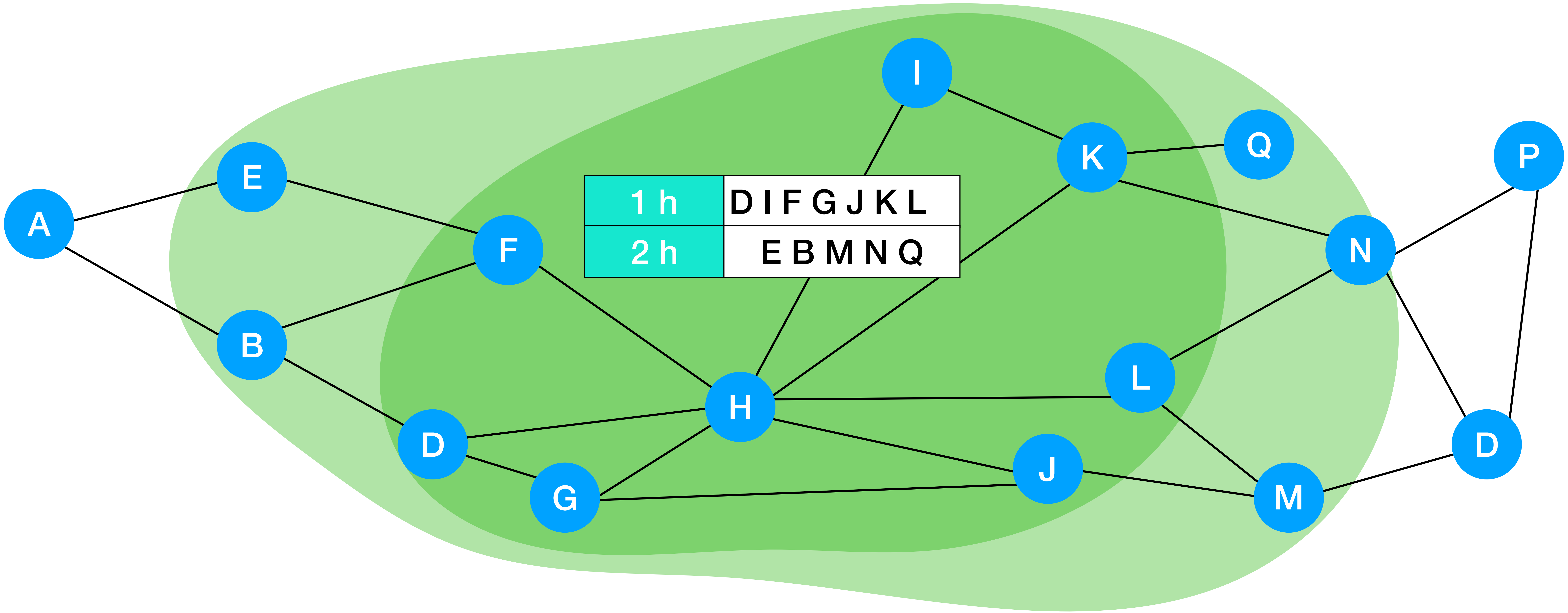
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



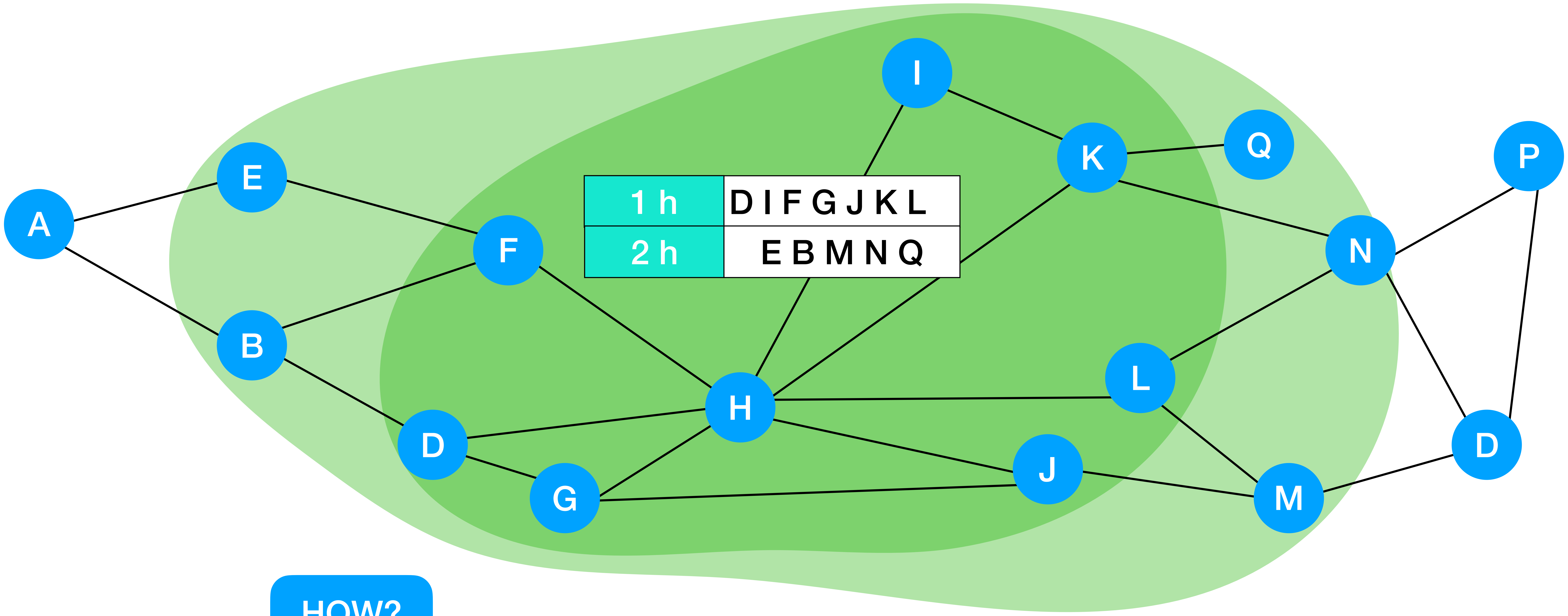
- Step 1: Each node discovers its two-hop neighbours proactively by sending “hello” packets with a list of the sender’s neighbour set

1 Optimised Link State Routing (OLSR) (2)



- Step 2: Each node selects a subset of its 1-hop neighbours to forward its link states (Multipoint Relays, MPRs). Each node maintains information about the set of neighbours that have selected it as MPR.

1 Optimised Link State Routing (OLSR) (2)



HOW?

- Step 2: Each node selects a subset of its 1-hop neighbours to forward its link states (Multipoint Relays, MPRs). Each node maintains information about the set of neighbours that have selected it as MPR.

1

MPR Selection Algorithm (1)

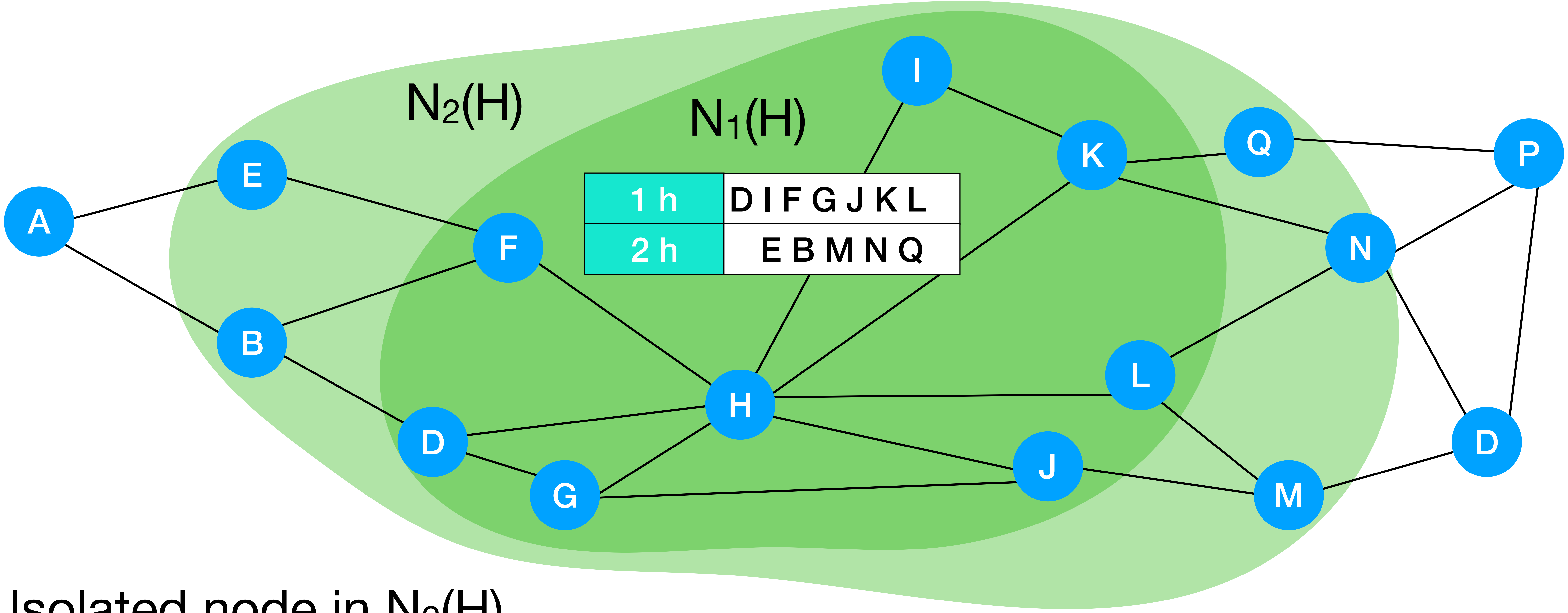
- **Goal**: given a node H with 1-hop neighbours $N_1(H)$ and 2-hop neighbours $N_2(H)$, select within $N_1(H)$ the smallest possible set of nodes that cover the entire set of nodes in $N_2(H)$.

- **Procedure**:
  1. Select all nodes in $N_1(H)$ which cover isolated nodes in $N_2(H)$, i.e., nodes only reachable by only one member of $N_1(H)$
  2. Amongst all unselected nodes in $N_1(H)$, select the node that covers the maximum number of uncovered nodes in $N_2(H)$
  3. Repeat Step 2 until all nodes are covered

- This is a heuristic approach, optimal MPR is NP-complete

1

MPR Selection Algorithm (2)



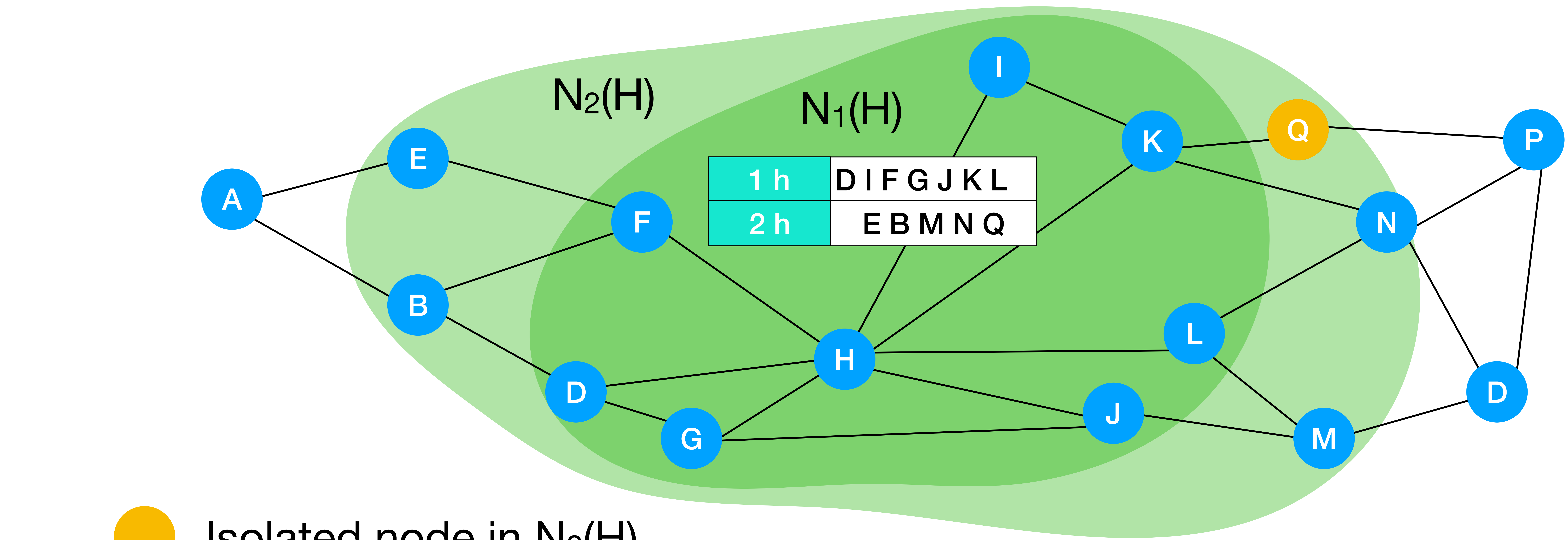
Isolated node in $N_2(H)$

Nodes in MPR(H)

Covered node in $N_2(H)$

Step 1: Search for isolated nodes in $N_2(H)$ and select as MPR the nodes in $N_1(H)$ that cover them

MPR Selection Algorithm (2)

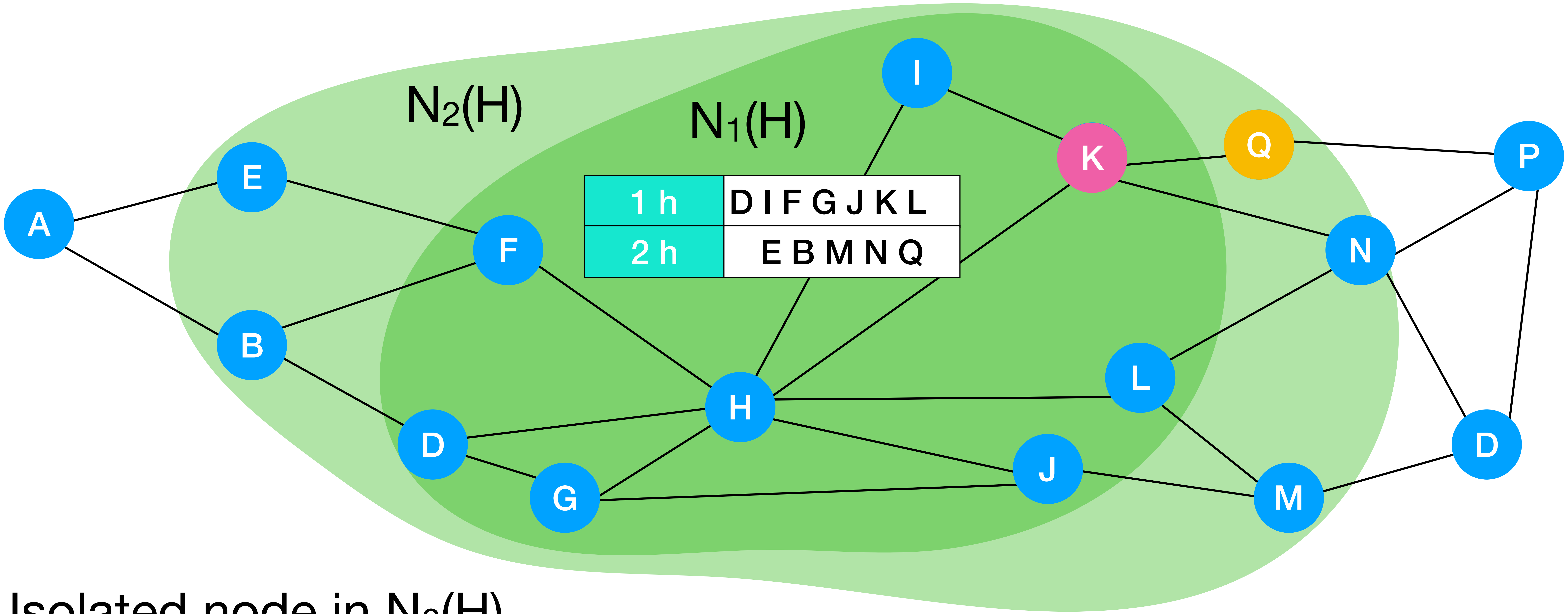


1 h	D I F G J K L
2 h	E B M N Q

- Isolated node in $N_2(H)$
- Nodes in MPR(H)
- Covered node in $N_2(H)$

Step 1: Search for isolated nodes in $N_2(H)$ and select as MPR the nodes in $N_1(H)$ that cover them

MPR Selection Algorithm (2)



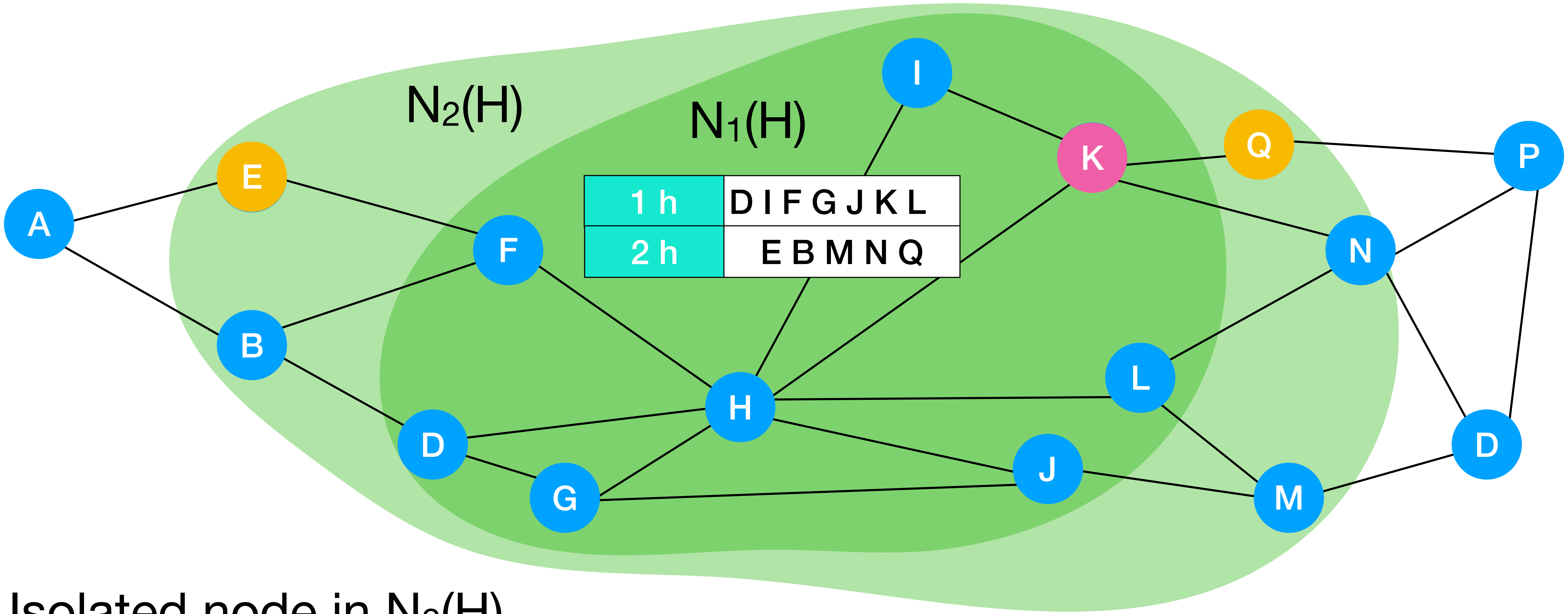
1 h	D I F G J K L
2 h	E B M N Q

- Isolated node in $N_2(H)$
- Nodes in MPR(H)
- Covered node in $N_2(H)$

Step 1: Search for isolated nodes in $N_2(H)$ and select as MPR the nodes in $N_1(H)$ that cover them

1

MPR Selection Algorithm (2)



Isolated node in $N_2(H)$

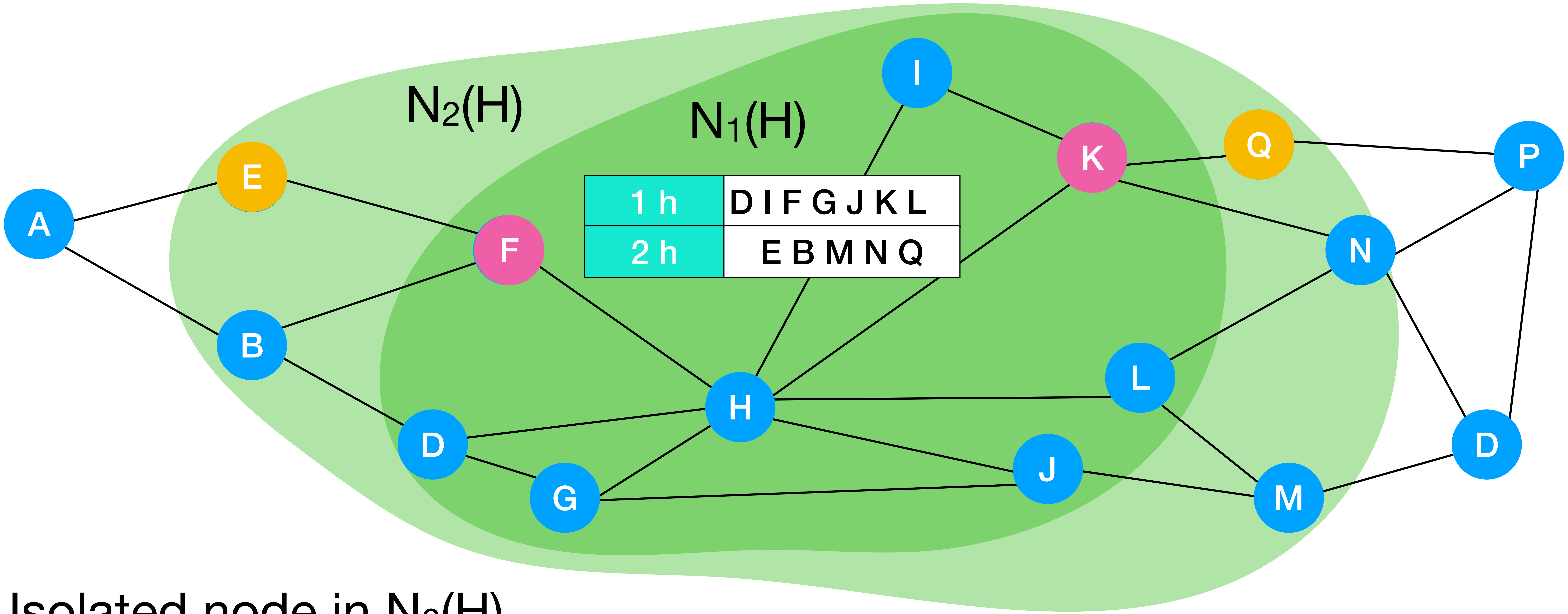
Nodes in MPR(H)

Covered node in $N_2(H)$

Step 1: Search for isolated nodes in $N_2(H)$ and select as MPR the nodes in $N_1(H)$ that cover them

MPR Selection Algorithm (2)

1



Isolated node in $N_2(H)$

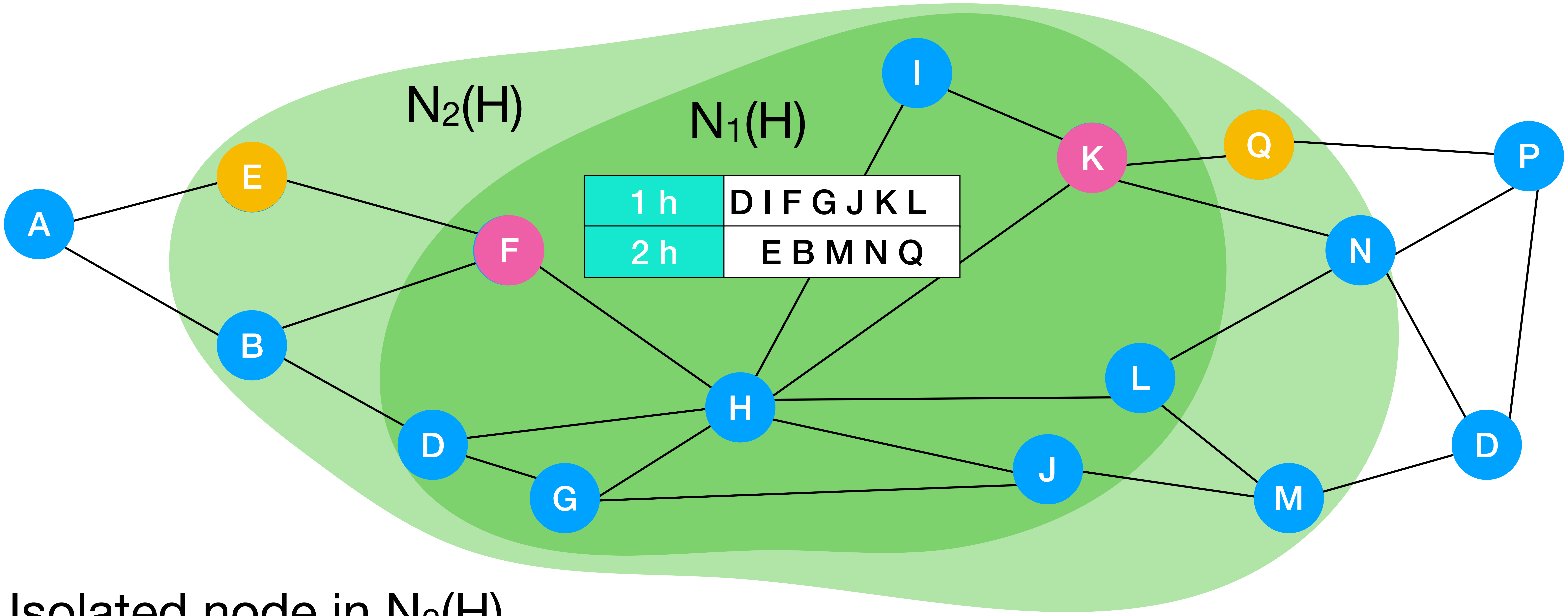
Nodes in MPR(H)

Covered node in $N_2(H)$

Step 1: Search for isolated nodes in $N_2(H)$ and select as MPR the nodes in $N_1(H)$ that cover them

1

MPR Selection Algorithm (2)



Isolated node in $N_2(H)$

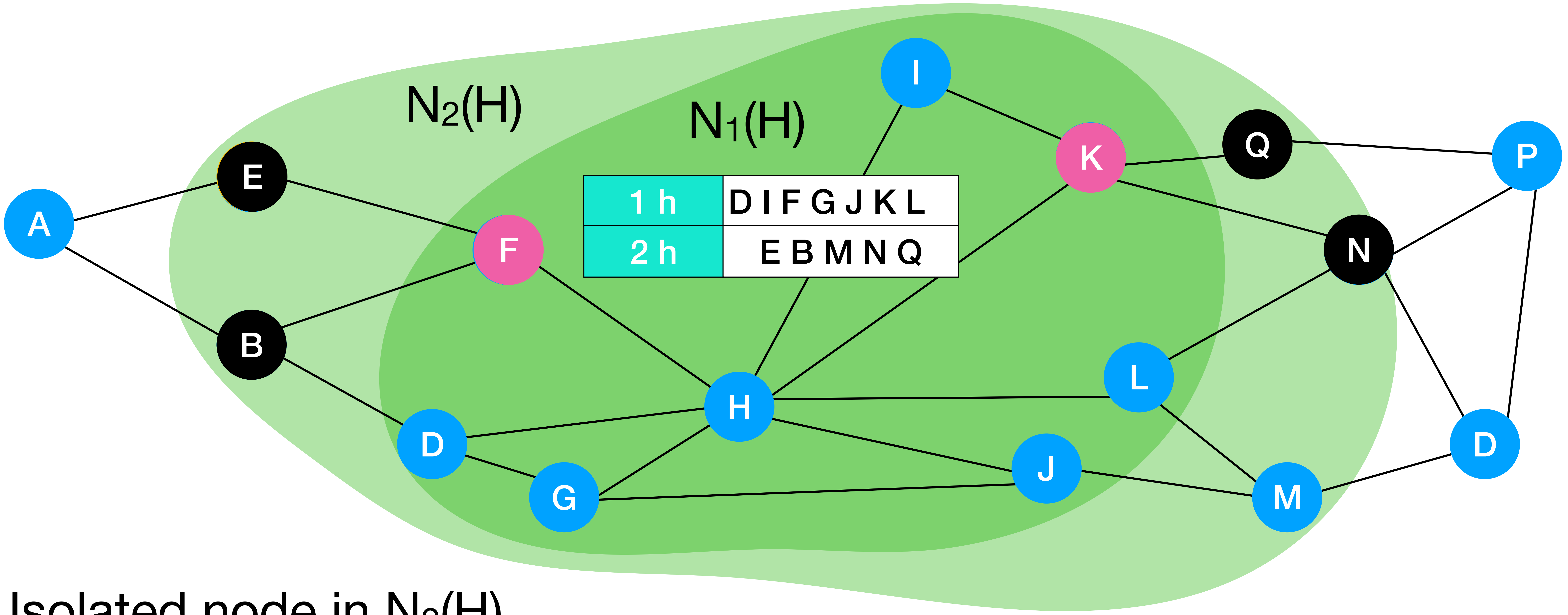
Nodes in MPR(H)

Covered node in $N_2(H)$

Step 2/3: See what notes in $N_2(H)$ are still uncovered and greedily choose the nodes in $N_1(H)$ that cover most of them.

1

MPR Selection Algorithm (2)



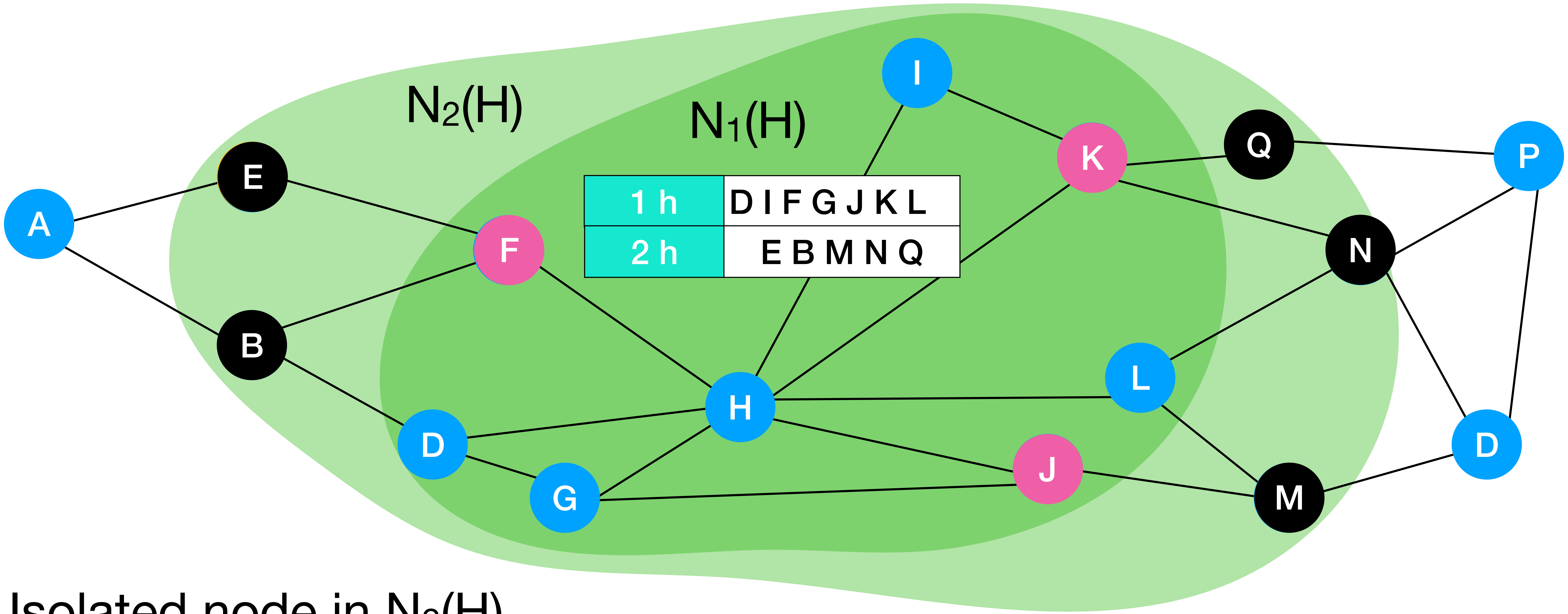
1 h	D I F G J K L
2 h	E B M N Q

- Isolated node in $N_2(H)$
- Nodes in MPR(H)
- Covered node in $N_2(H)$

Step 2/3: See what notes in $N_2(H)$ are still uncovered and greedily choose the nodes in $N_1(H)$ that cover most of them.

1

MPR Selection Algorithm (2)



1 h	D I F G J K L
2 h	E B M N Q

Isolated node in $N_2(H)$

Nodes in MPR(H)

Covered node in $N_2(H)$

Step 2/3: See what notes in $N_2(H)$ are still uncovered and greedily choose the nodes in $N_1(H)$ that cover most of them.

1

Optimised Link State Routing (OLSR) - forwarding

- Once each node selected its MPRs, the topology database is built as in regular link-state routing.
- Having done that, nodes perform **forwarding** by running Dijkstra, store shortest-path next-hop for each destination, **through their MPRs**.
- Observe that nodes tend to select highly connected nodes as MPRs, i.e., some nodes are MPRs of several nodes.

✅ only a small subset of the network needs to be awake at any point in time, reduces routing table size, allows more nodes to sleep

❌ Asymmetric battery usage: MPRs are awake most of the time and their batteries drain faster.

1

Optimised Link State Routing (OLSR) - downsides

- Maintains routes to all nodes, all the time
  - good if most nodes need to talk to each other
  - causes overhead if communication is more sparse/rare
  - reveals entire topology to all nodes (bad for privacy)

- Solution: what if we build routes "on demand" just when we need them? (Reactive routing)

- (Want to know more about OLSR? Have fun reading the RFC 3626: https://www.rfc-editor.org/rfc/rfc3626.html)

2 Dynamic Source Routing (DSR) (1)

- **Reactive protocol**: no proactive network discovery when the network is initialised. Nodes perform a discovery process only when data needs to be sent.

- Route to a destination is discovered and stored by the source node and **embedded in the data packet**.

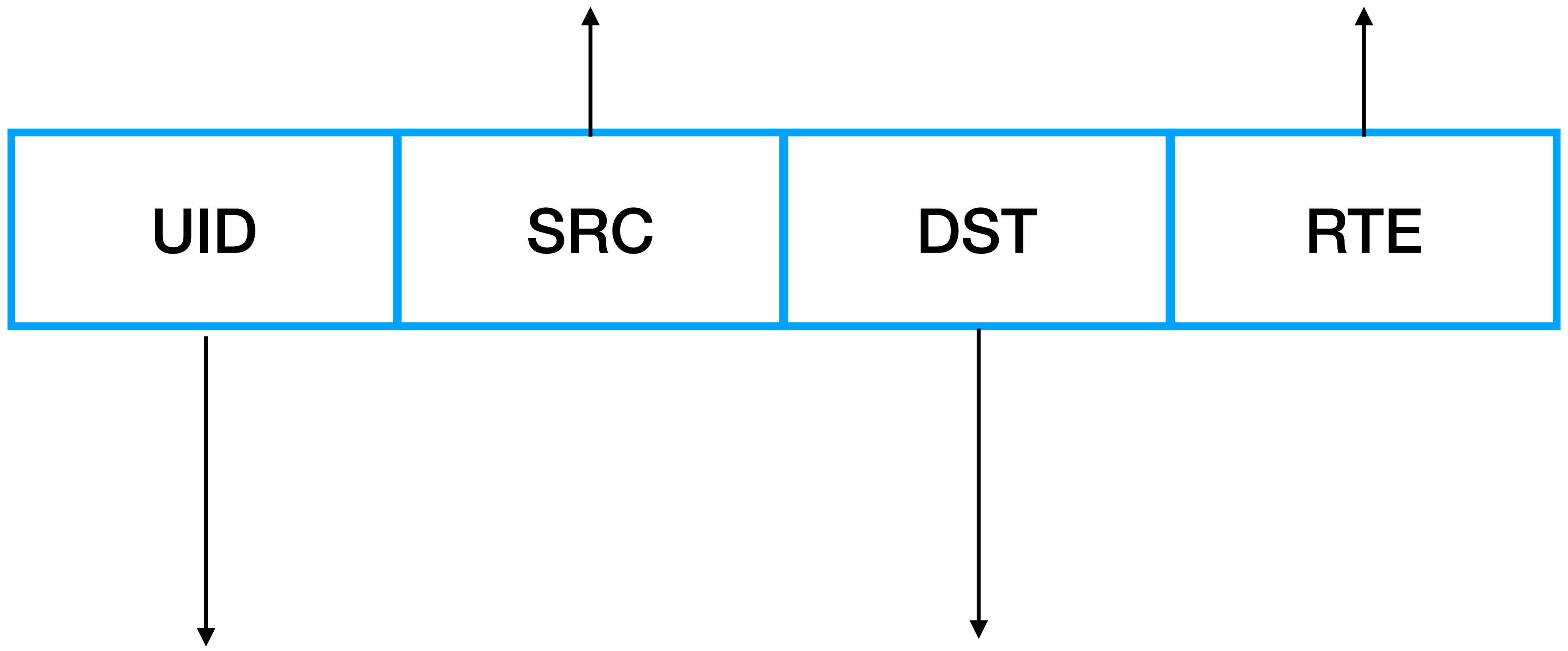
- Main idea:
  -when a source node generates data to send to a destination node of which it does not know the route to, it **floods** a **route request message (RREQ)**. Intermediate nodes append their ID.
  - when the RREQ reaches the destination, it sends back a **route reply message (RREP)** along reverse of path contained in RREQ.
  - when the source gets the RREP, it sends the data along the path written in the header of the RREP (called “source route”)

2 Dynamic Source Routing- RREQ (1)

Source. Node sending the RREQ

Route. source route learned by the RREQ

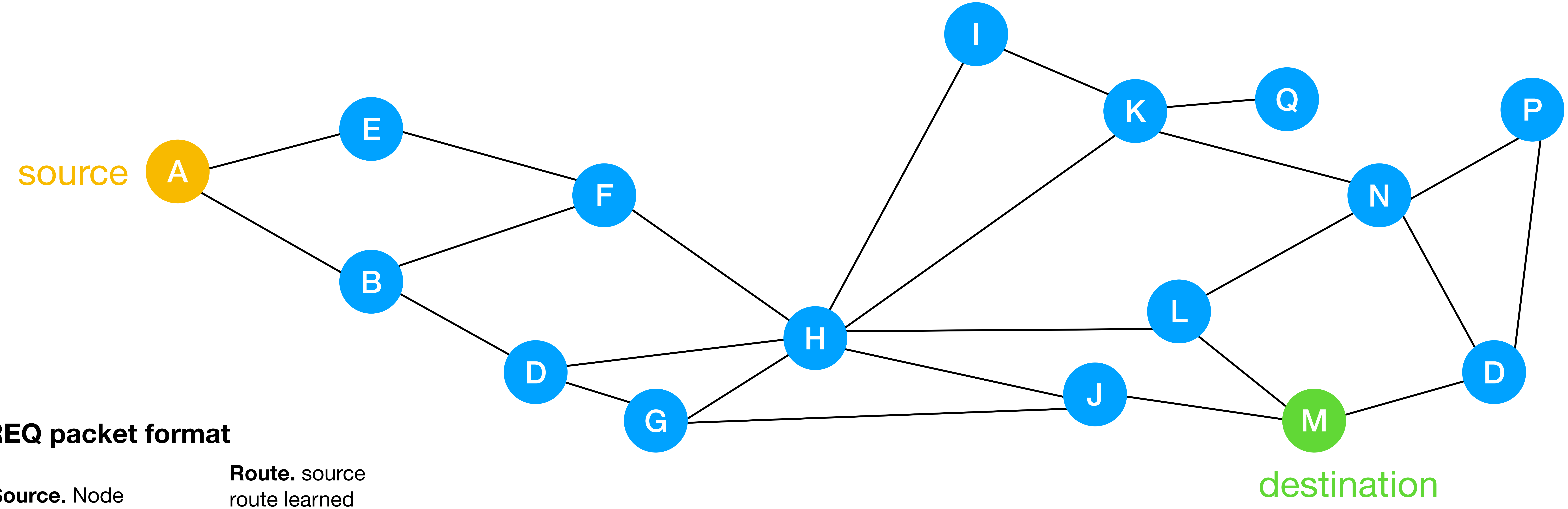
RREQ packet format



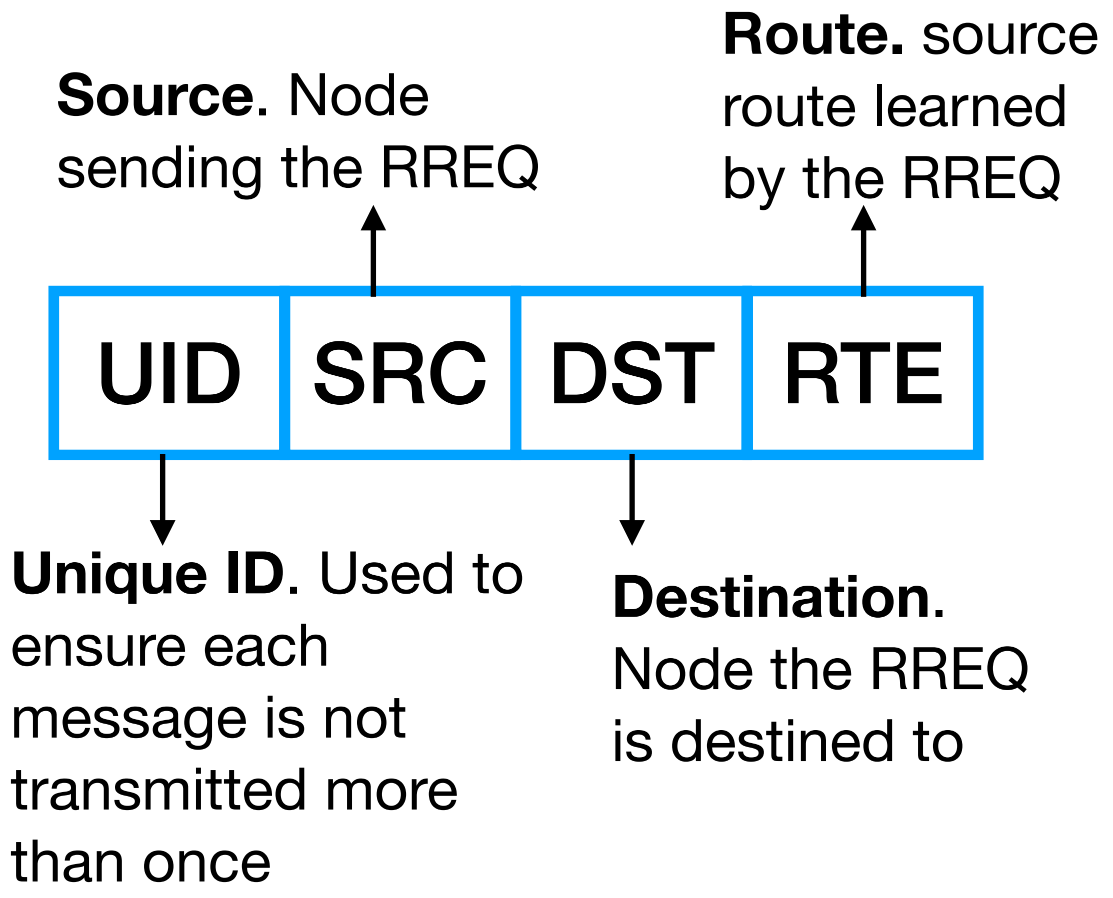
Unique ID. Used to ensure each message is not transmitted more than once

Destination. Node the RREQ is destined to

2 Dynamic Source Routing- RREQ (2)

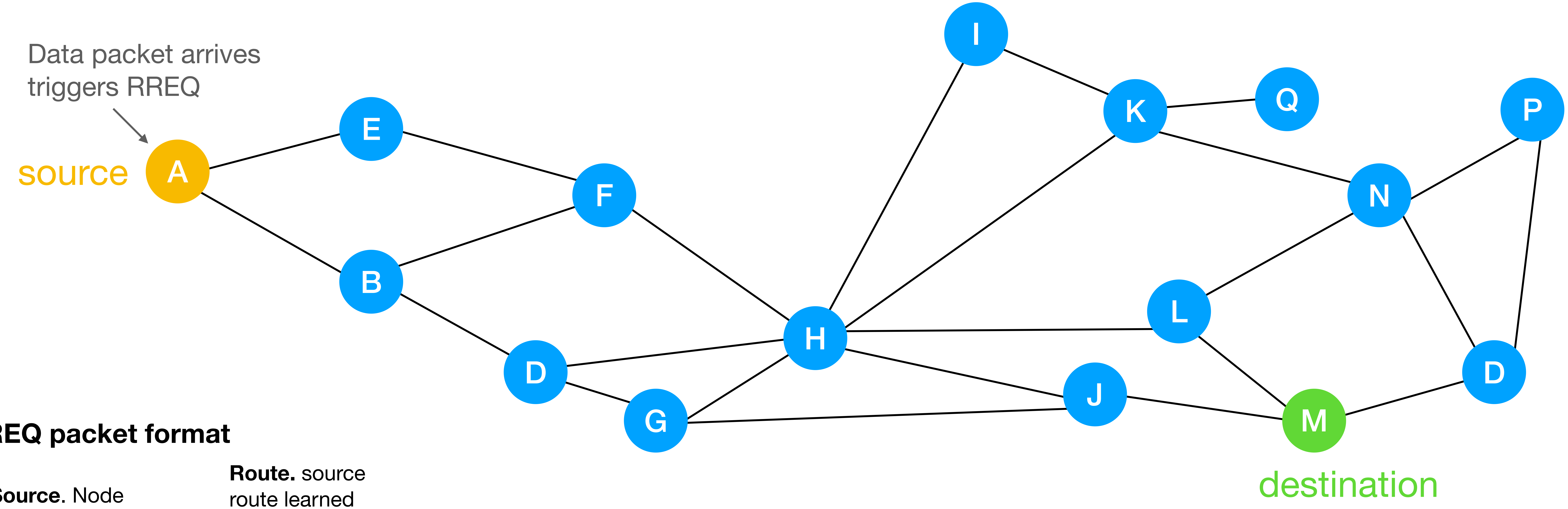


RREQ packet format

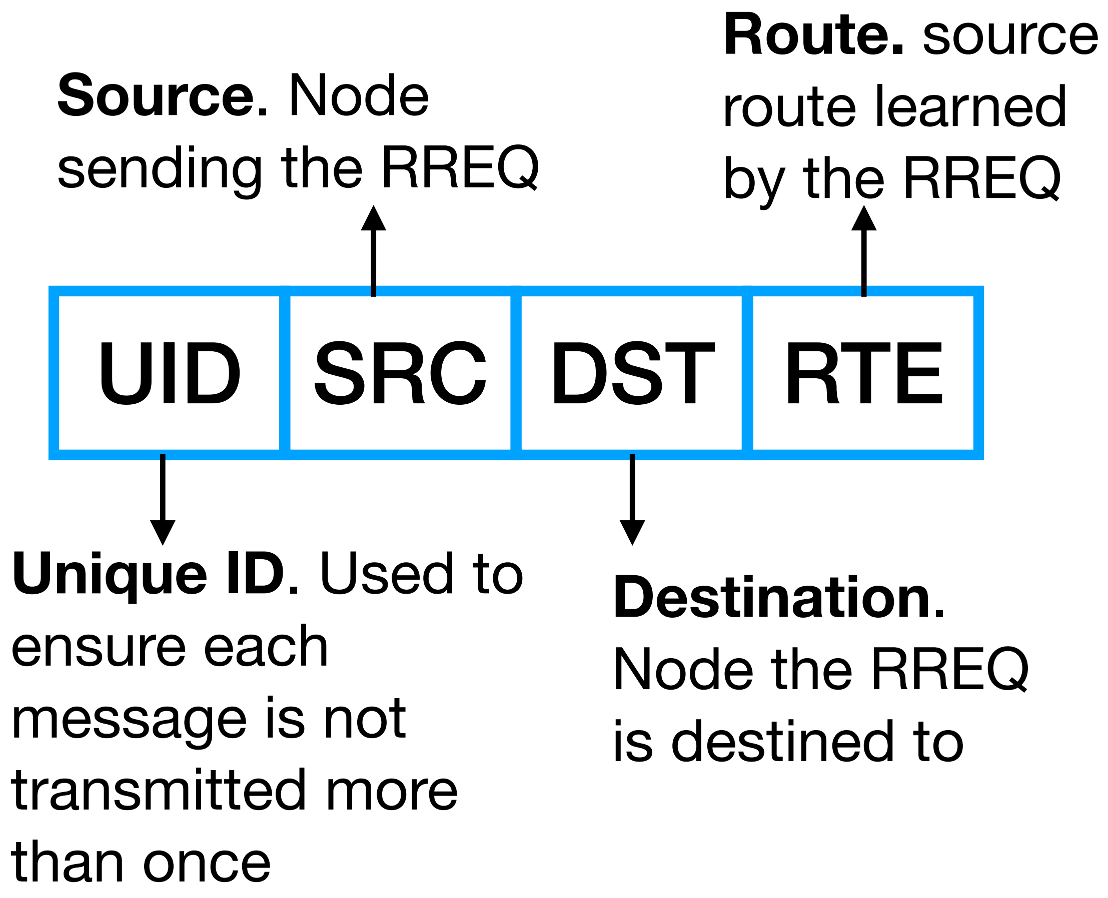


Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

② Dynamic Source Routing- RREQ (2)

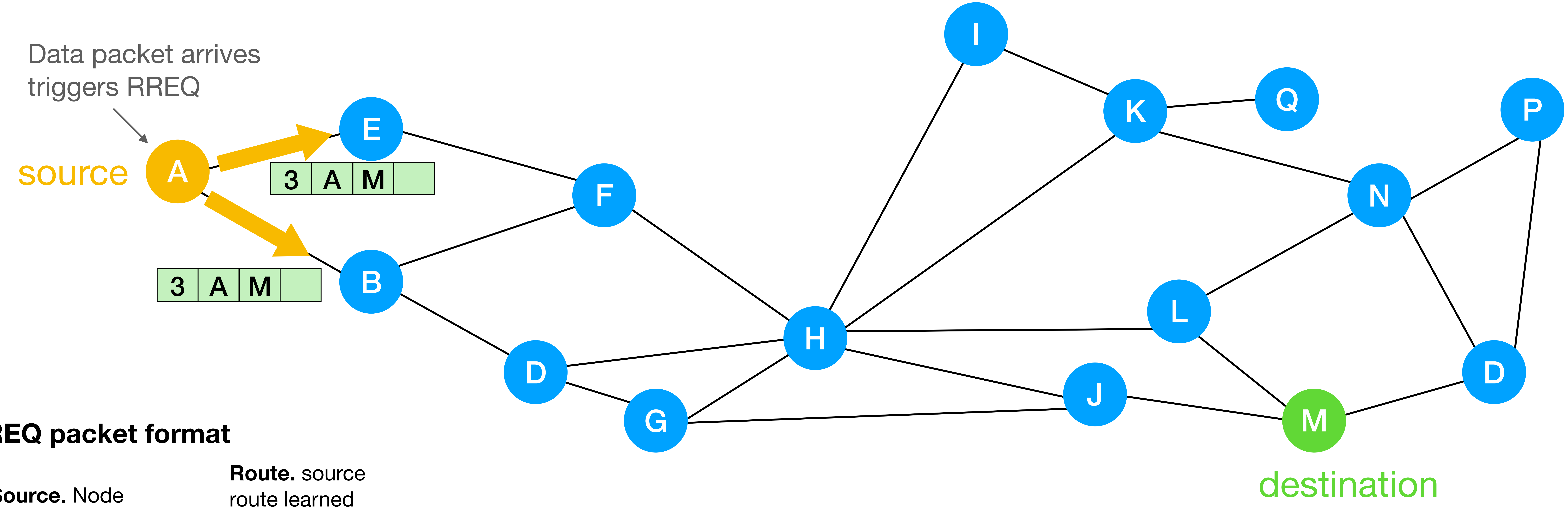


RREQ packet format

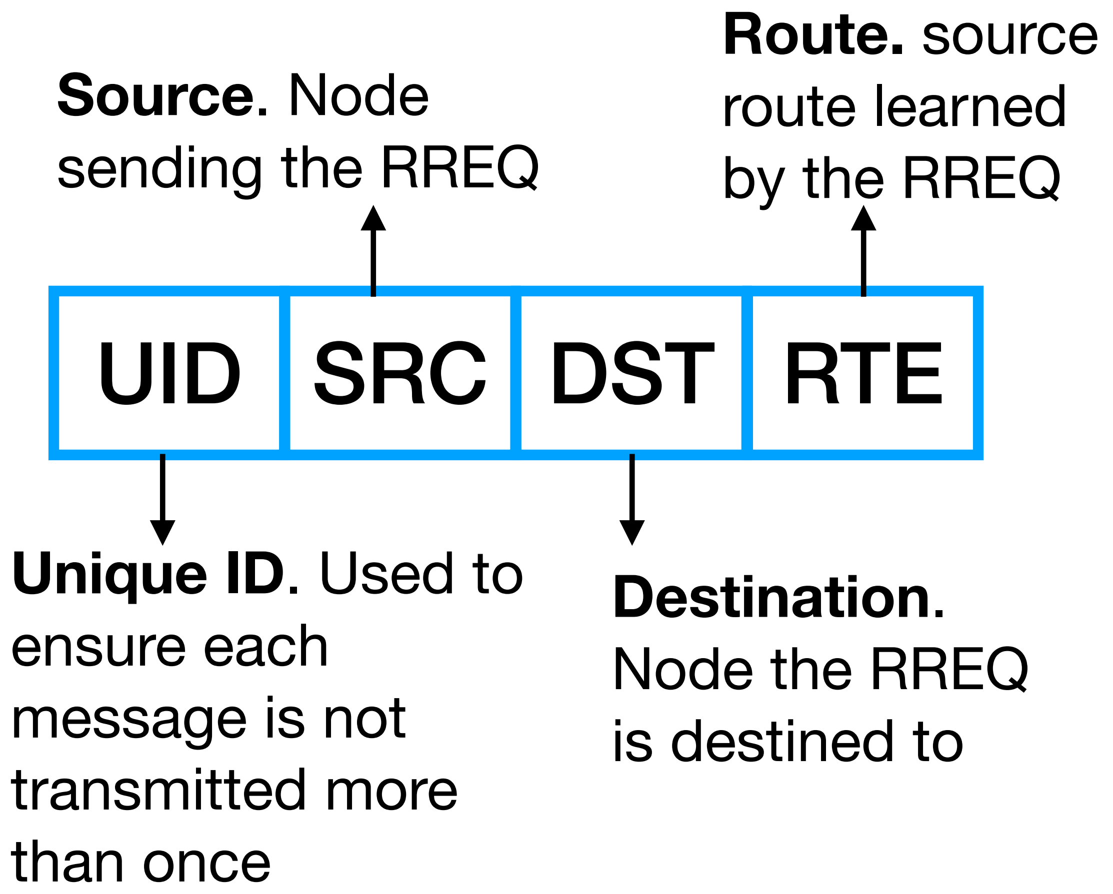


Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



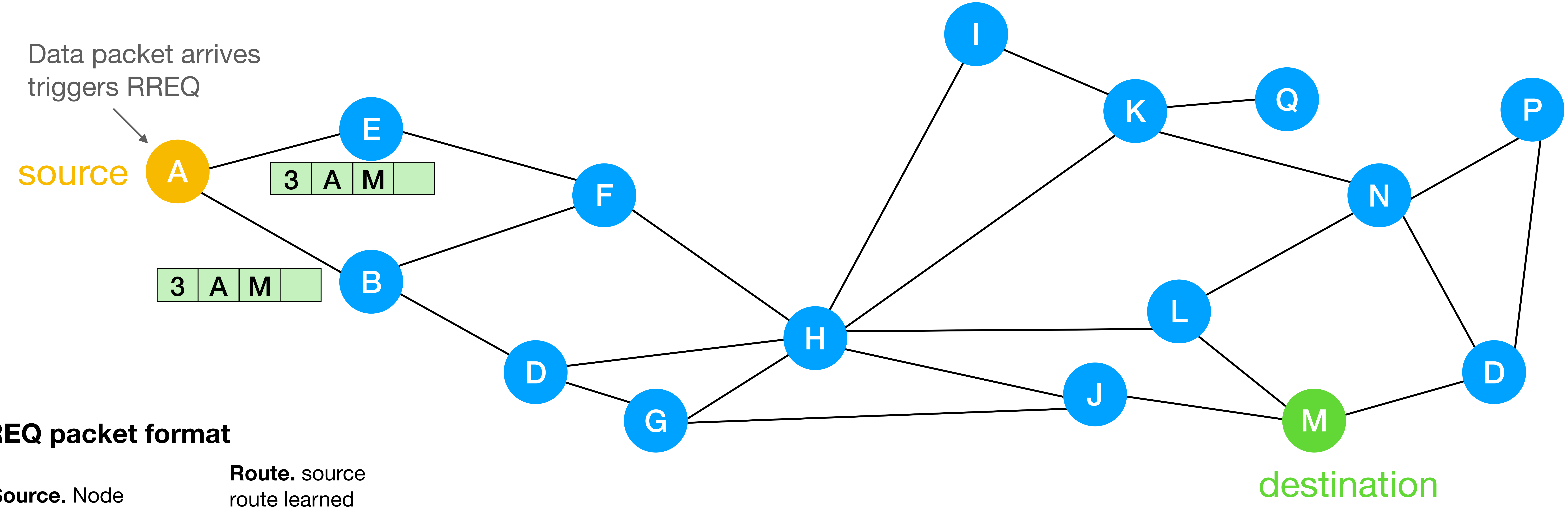
RREQ packet format



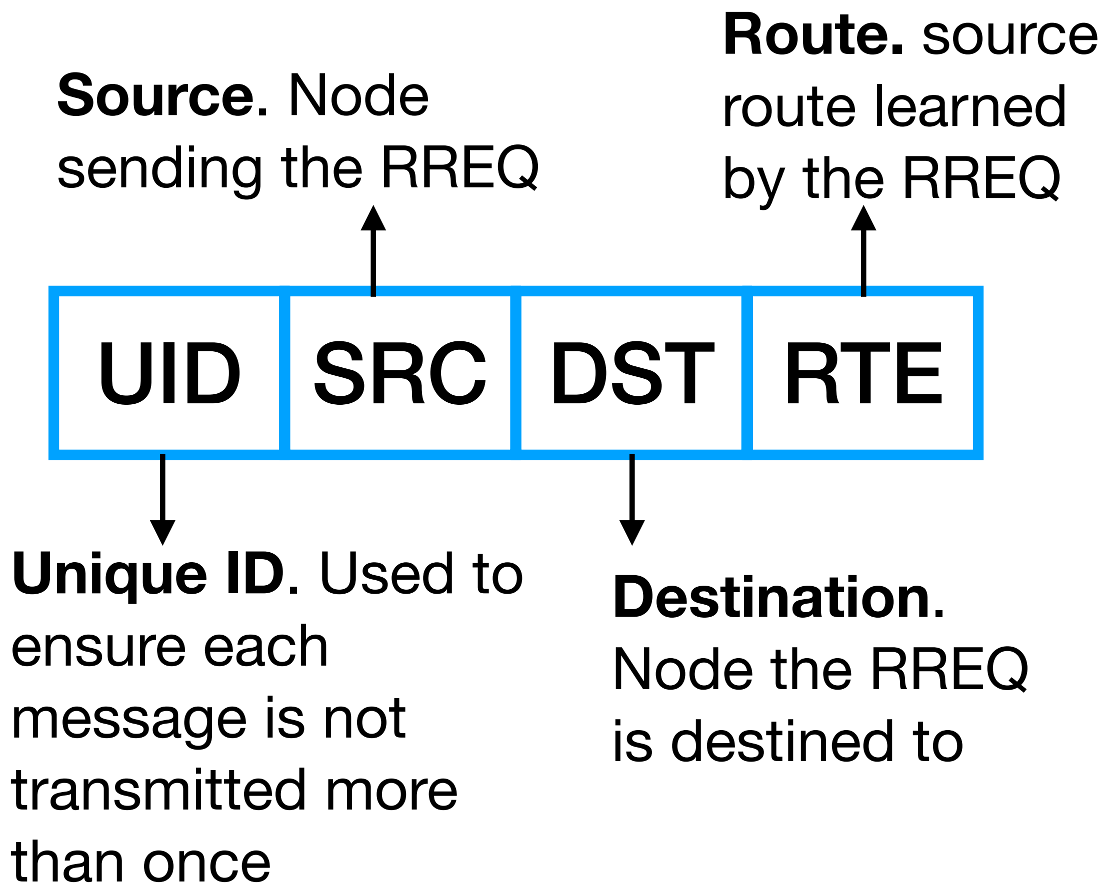
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



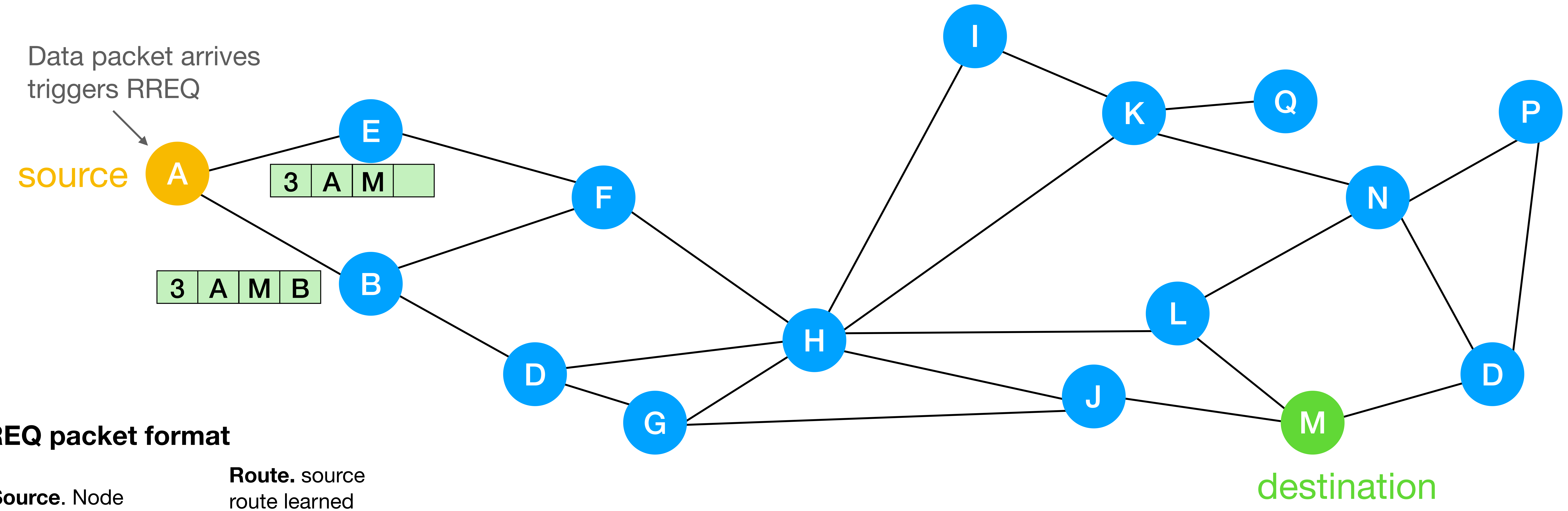
RREQ packet format



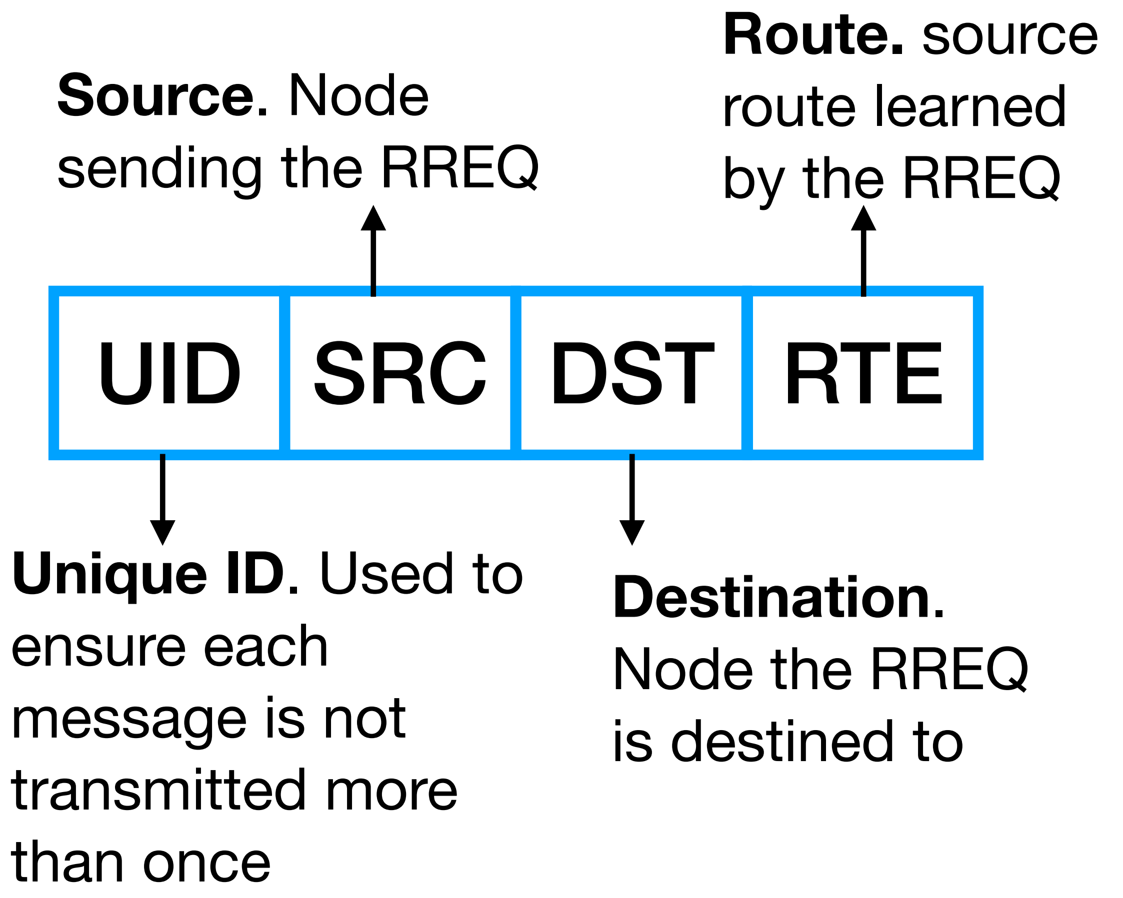
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



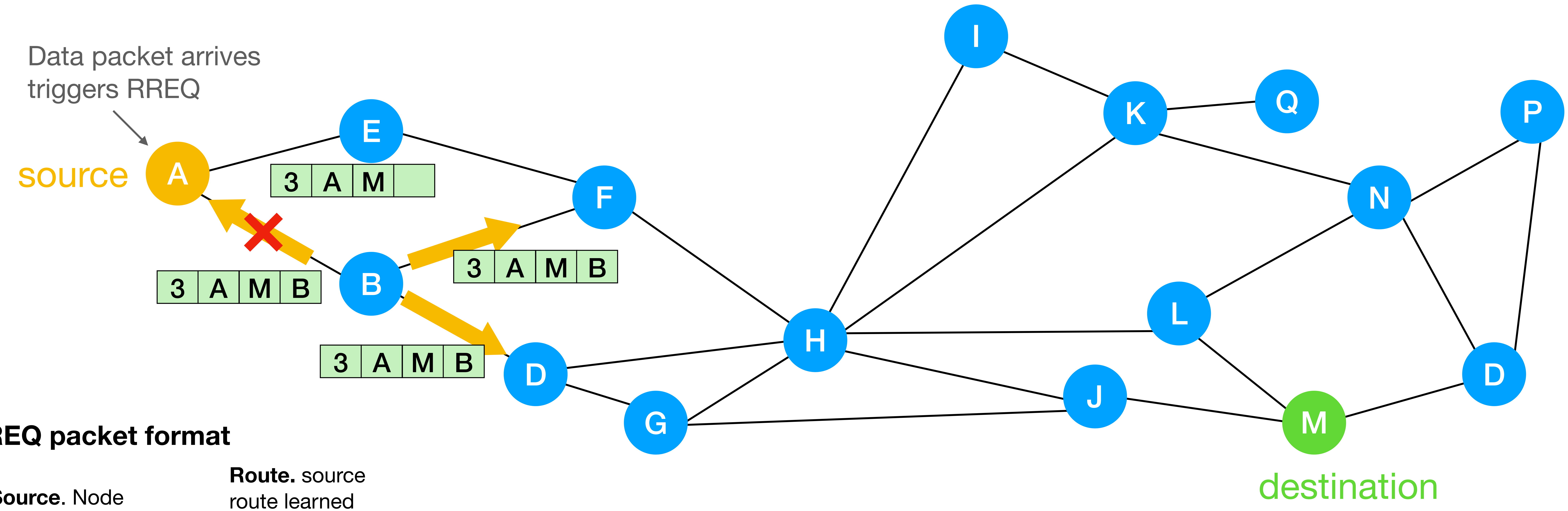
RREQ packet format



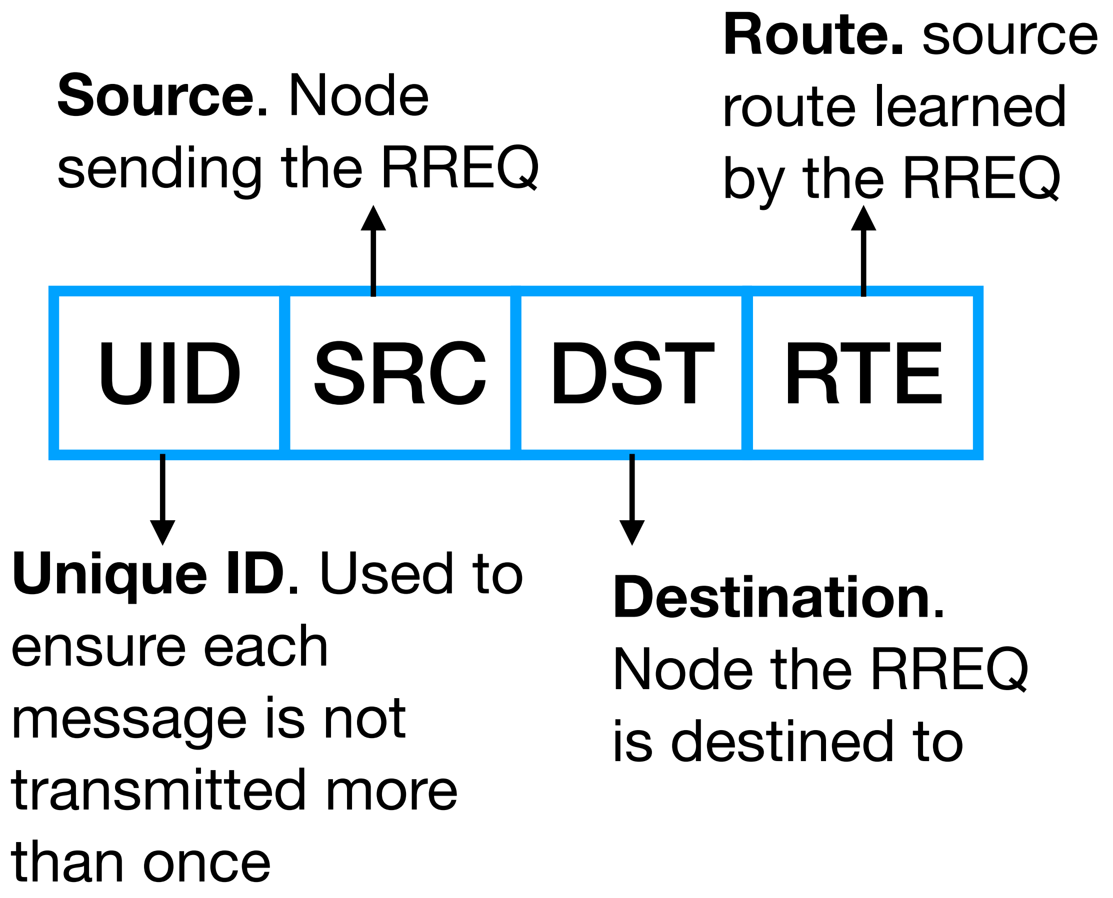
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



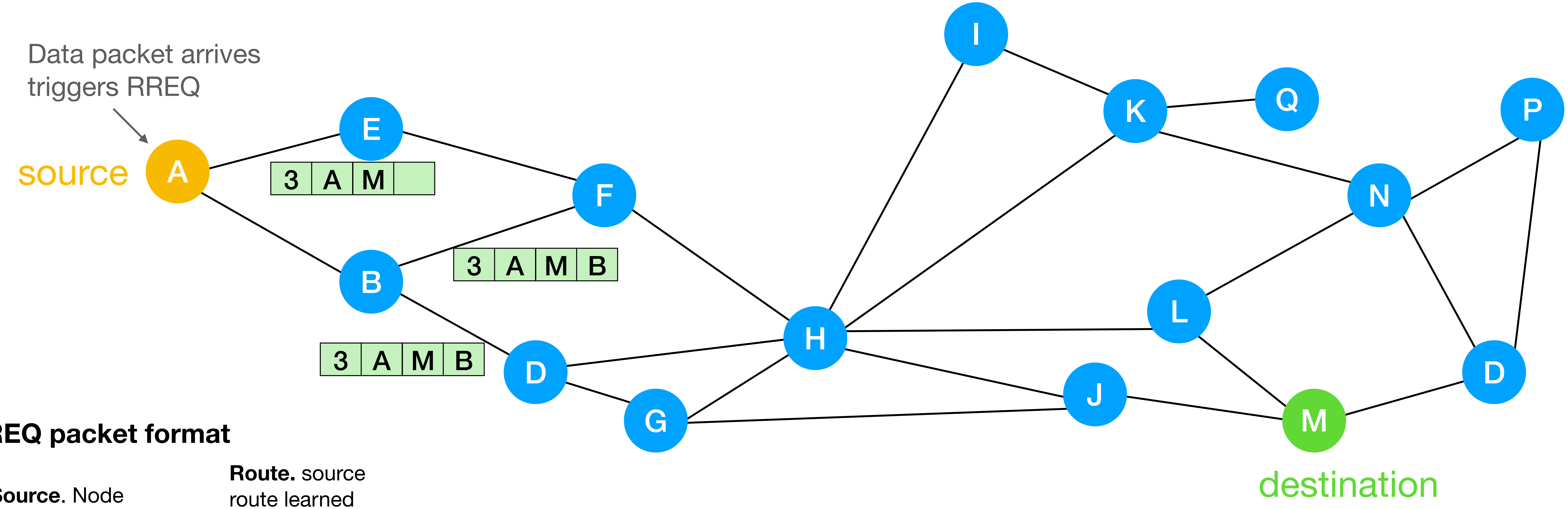
RREQ packet format



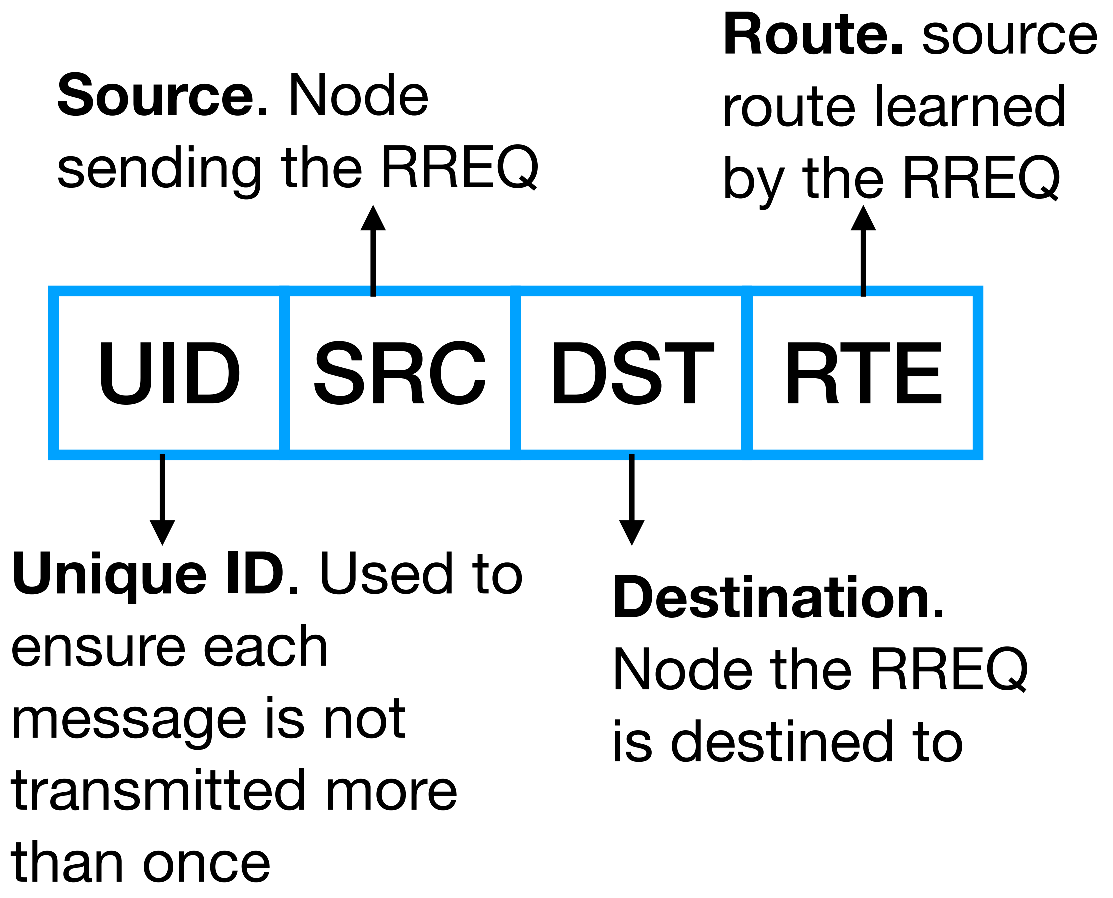
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

② Dynamic Source Routing- RREQ (2)



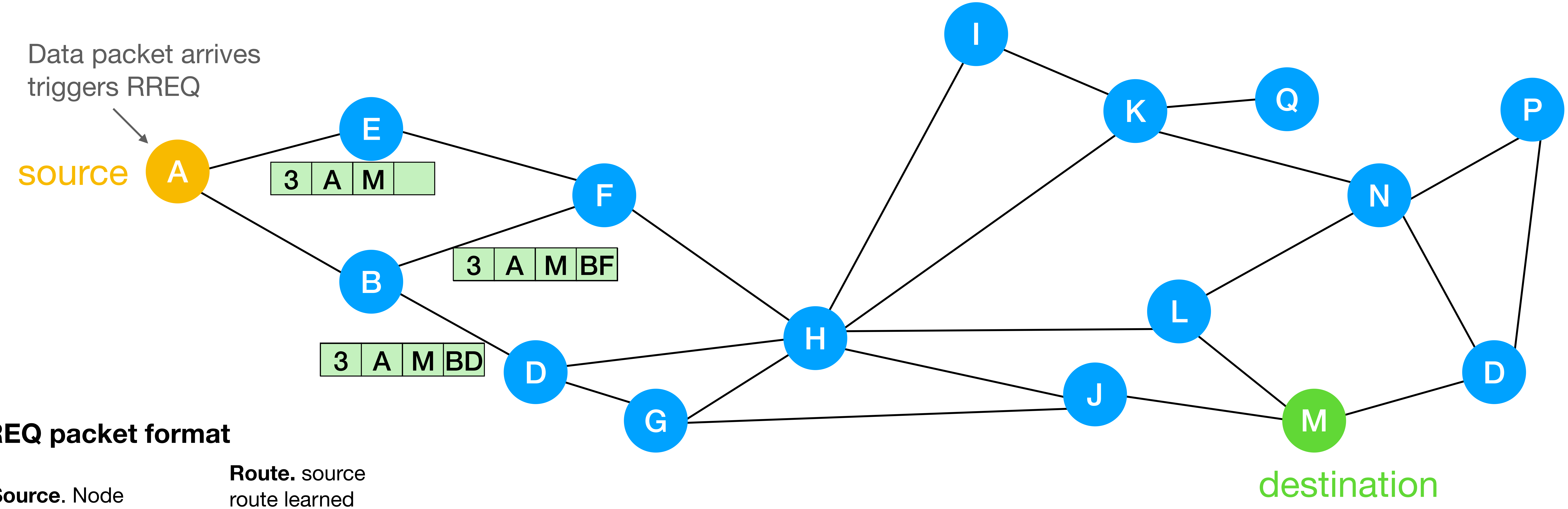
RREQ packet format



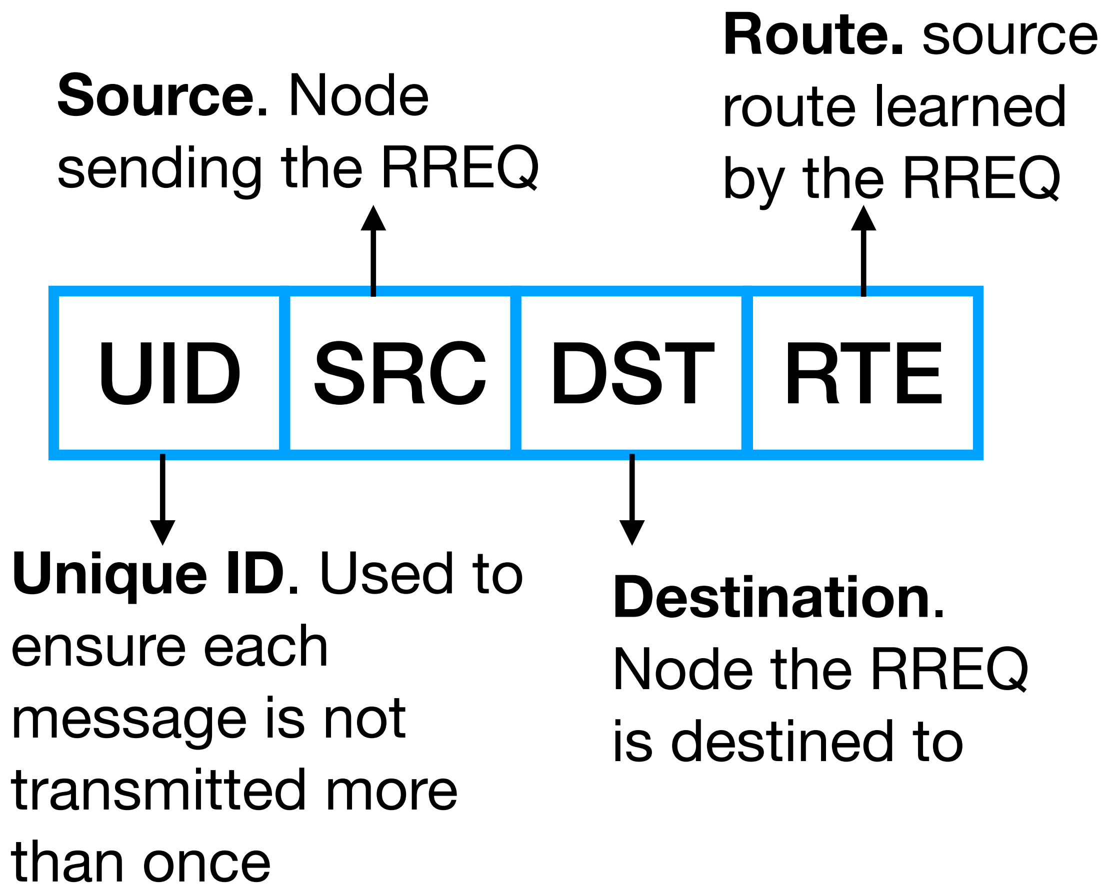
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)

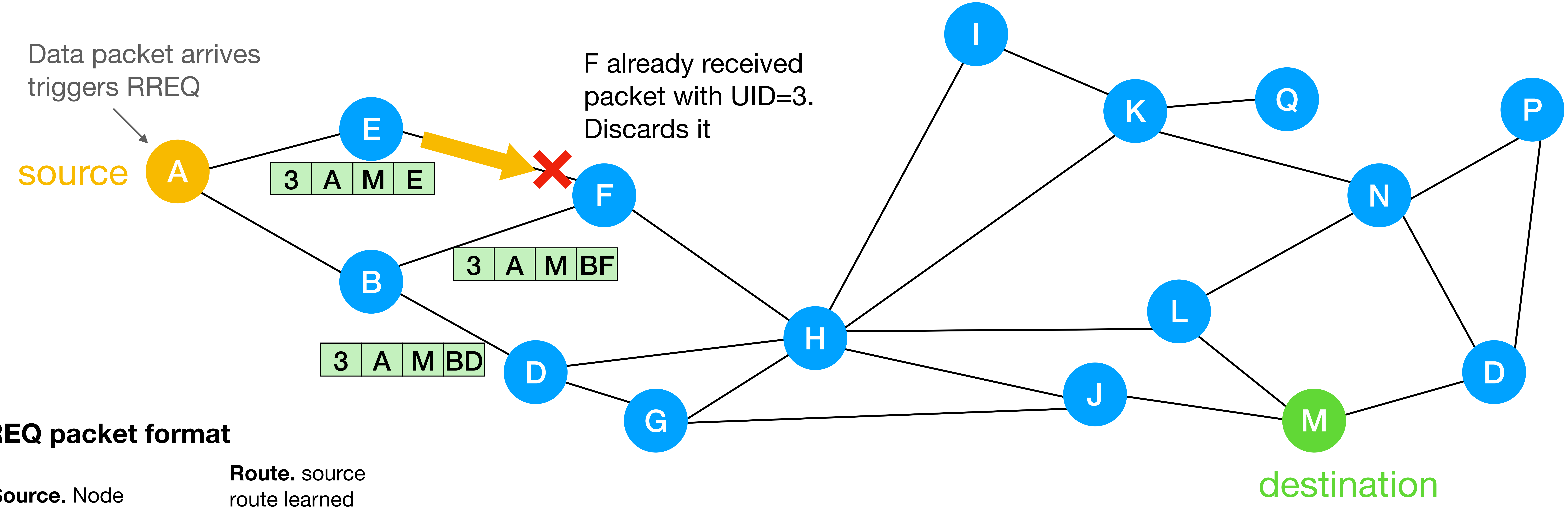


RREQ packet format

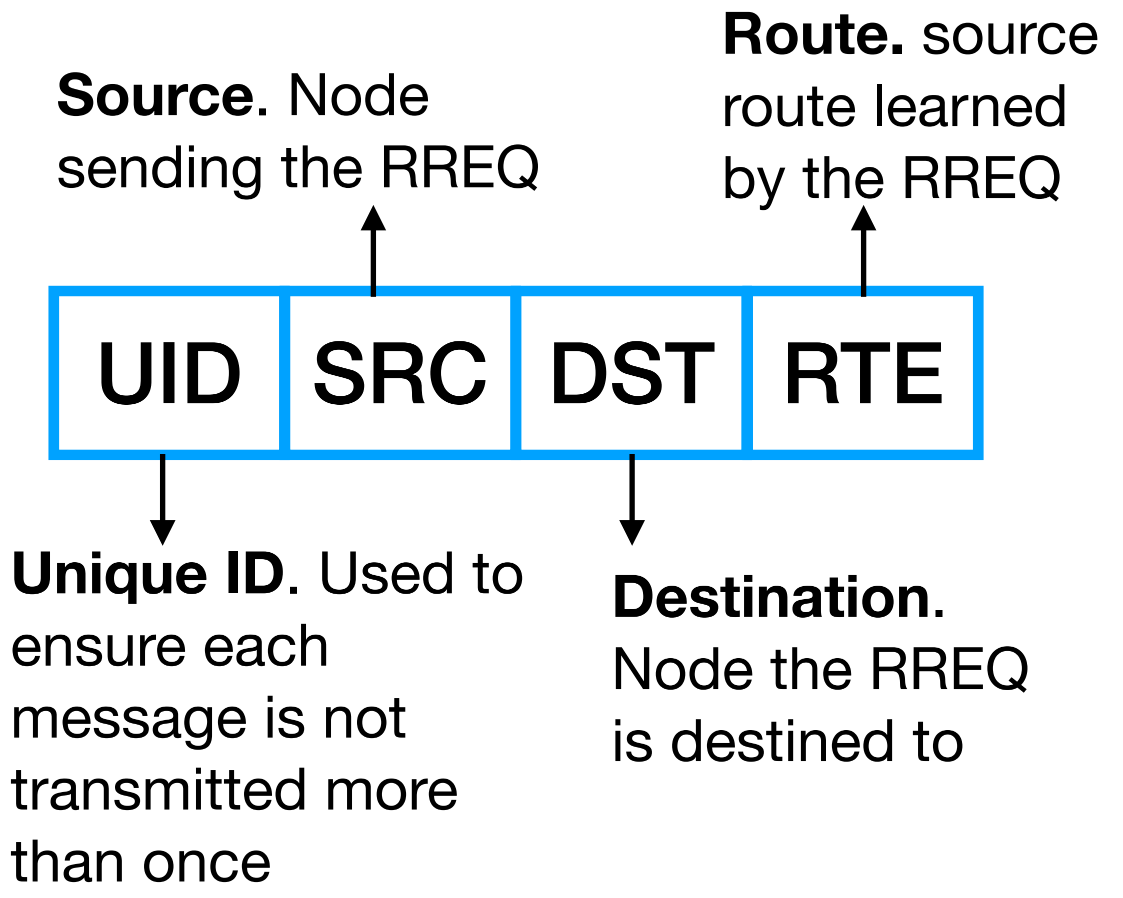


Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination.
Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

② Dynamic Source Routing- RREQ (2)



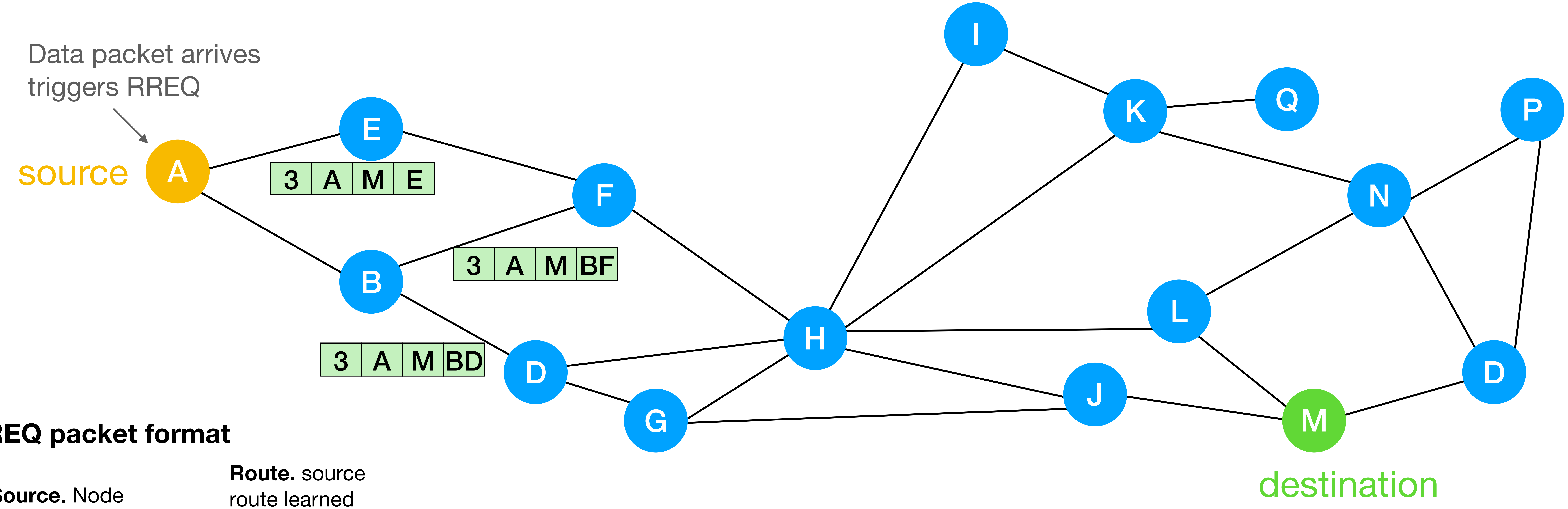
RREQ packet format



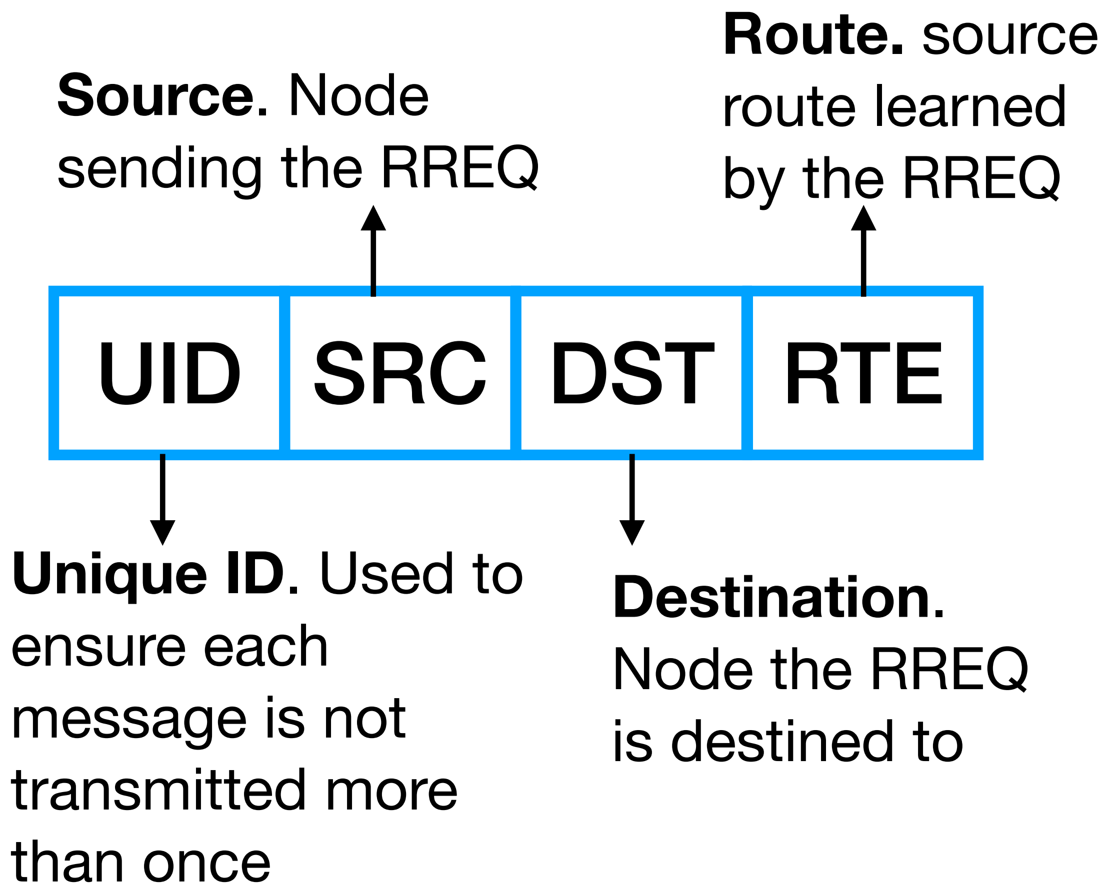
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



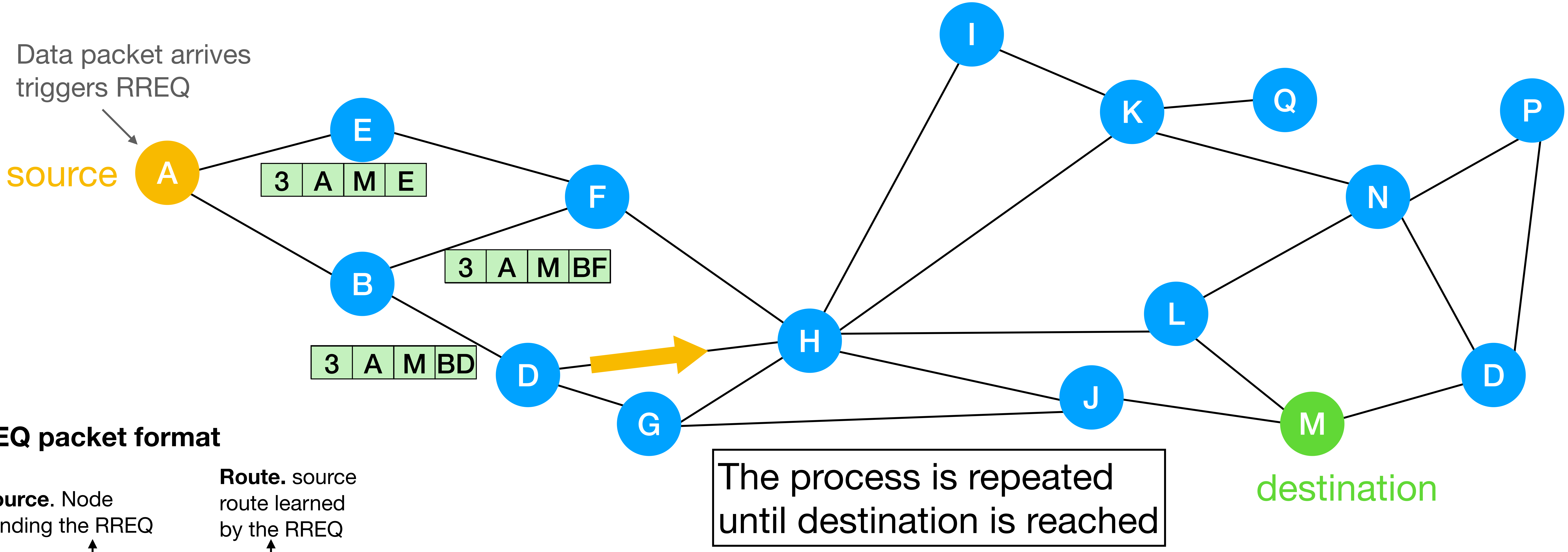
RREQ packet format



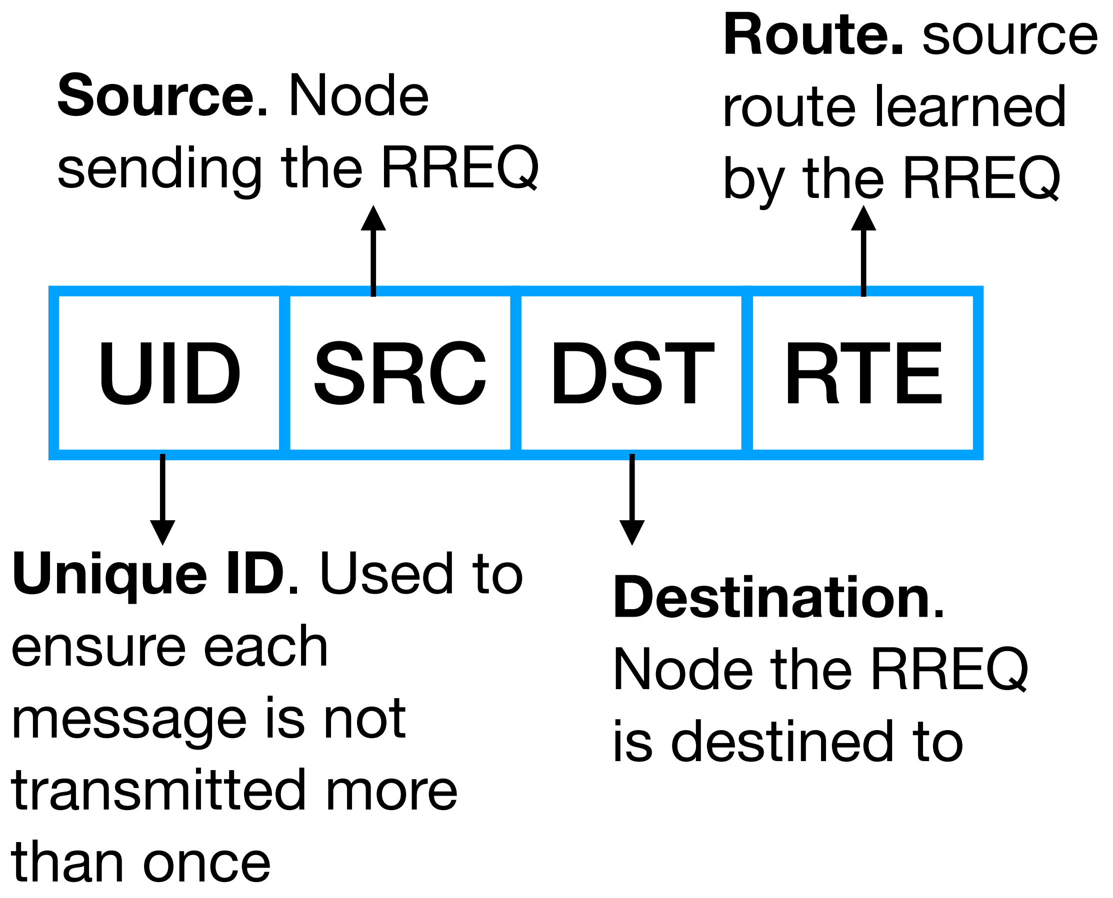
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)

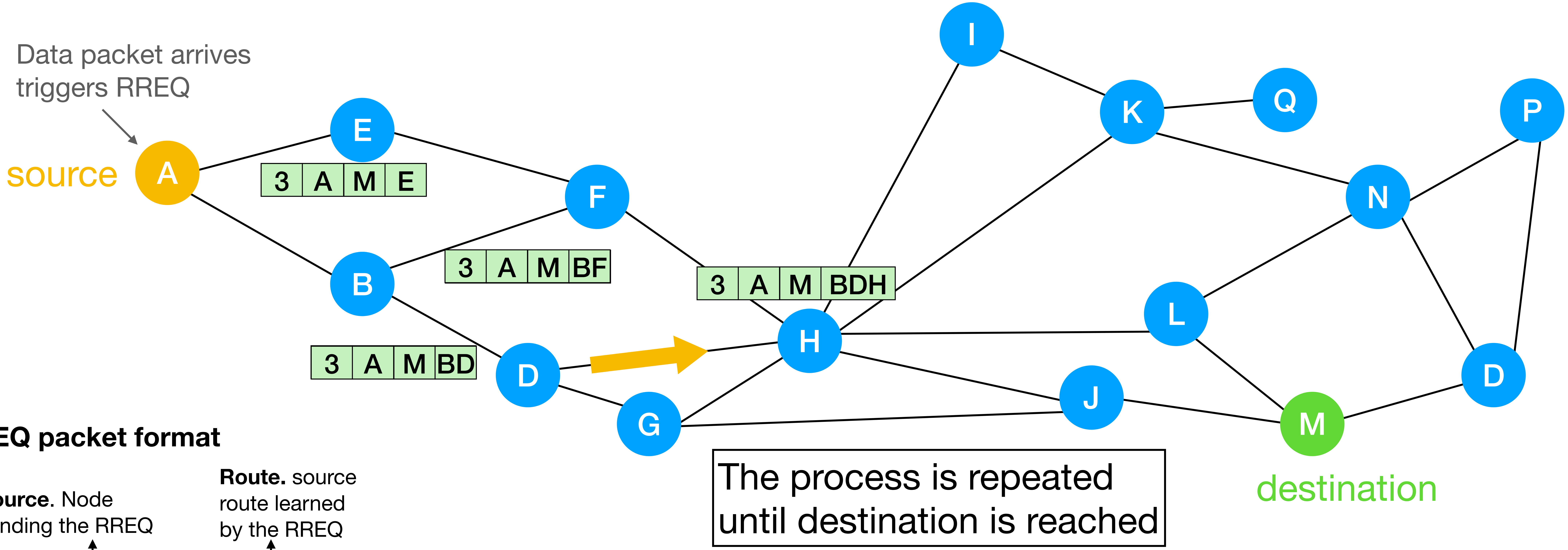


RREQ packet format

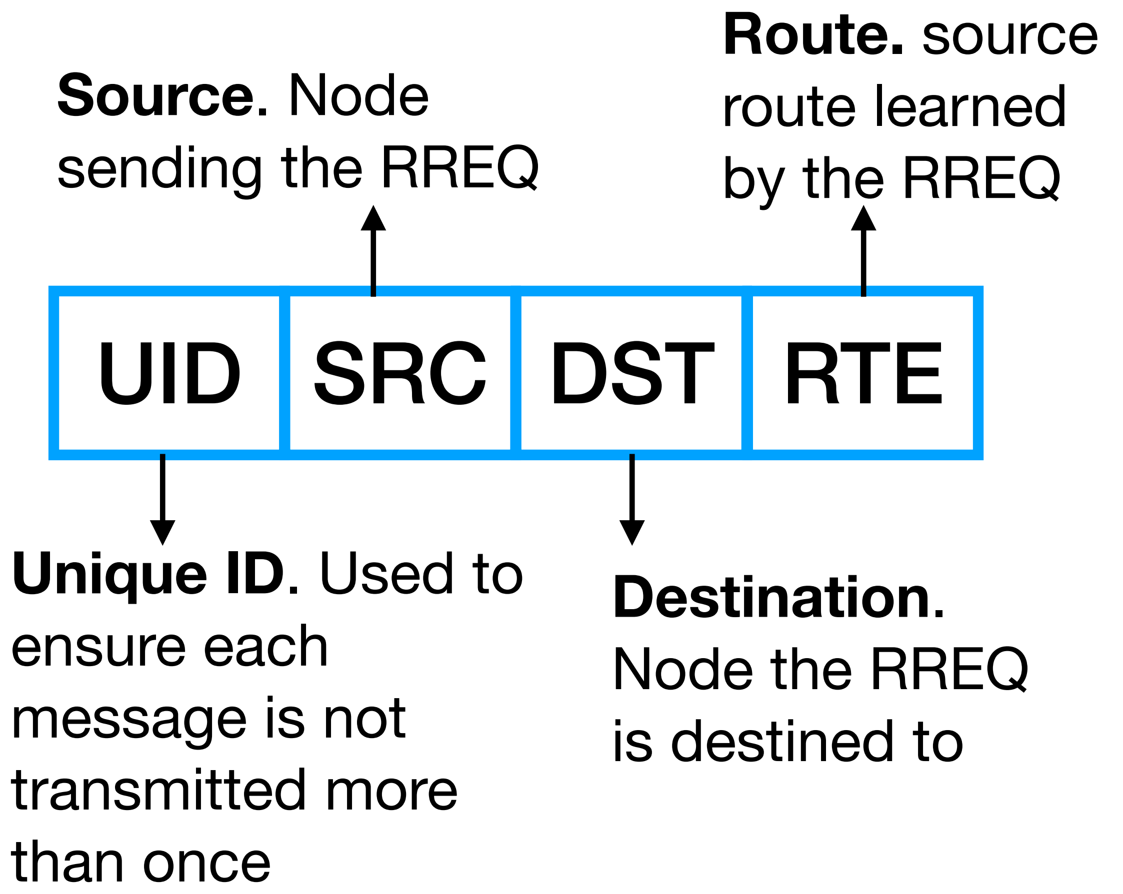


Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination.
Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

Dynamic Source Routing- RREQ (2)

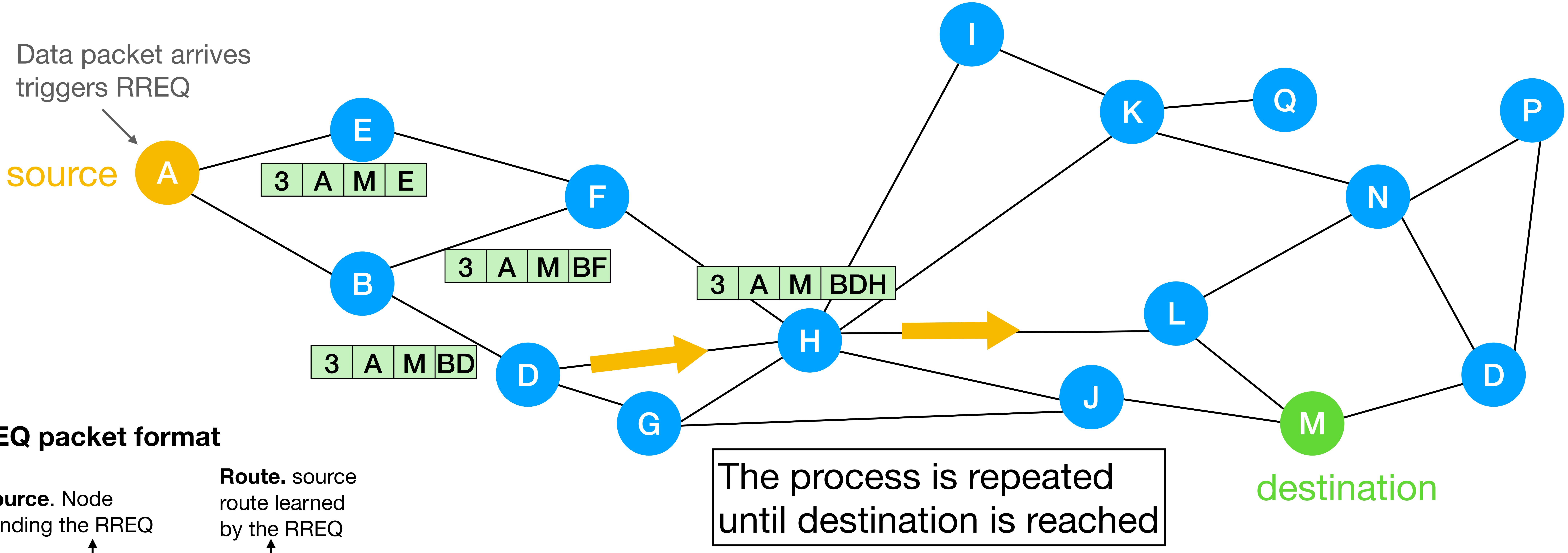


RREQ packet format

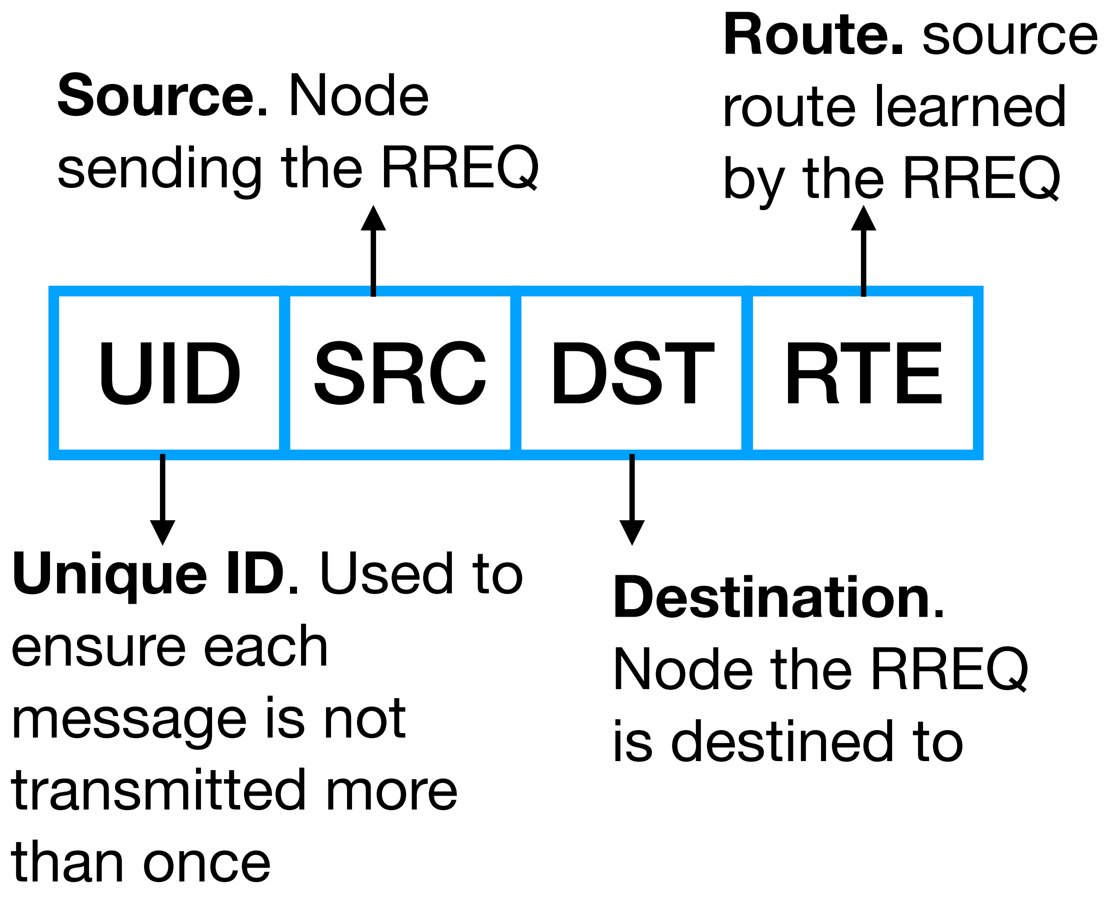


Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)

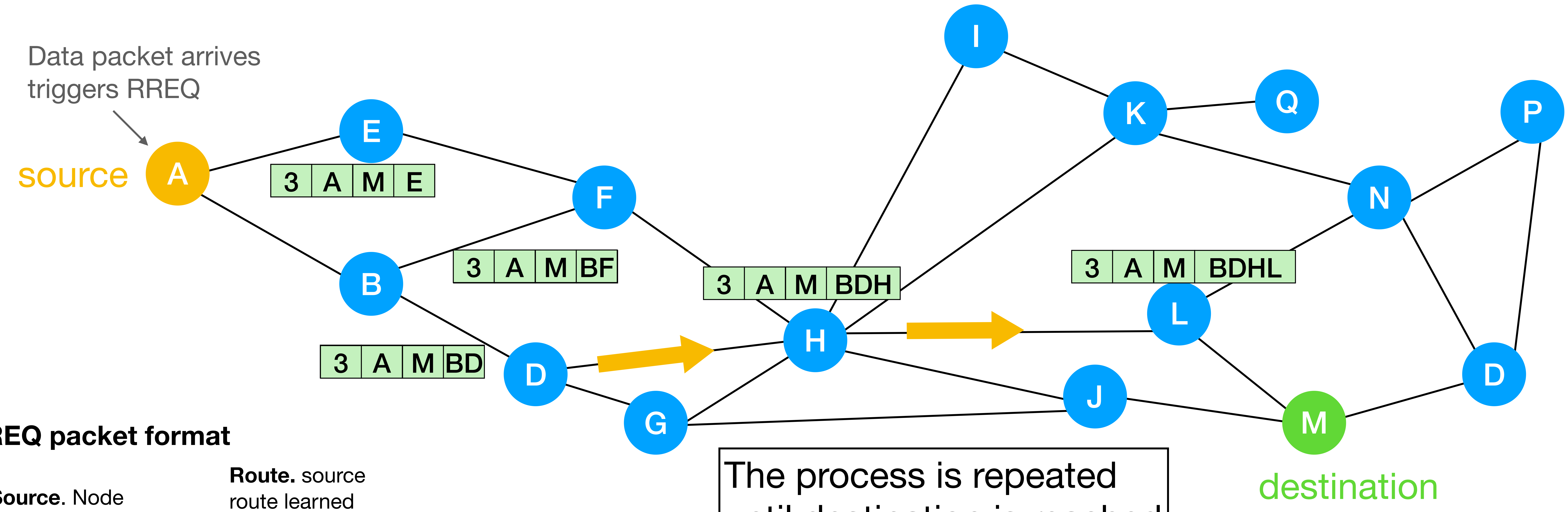


RREQ packet format



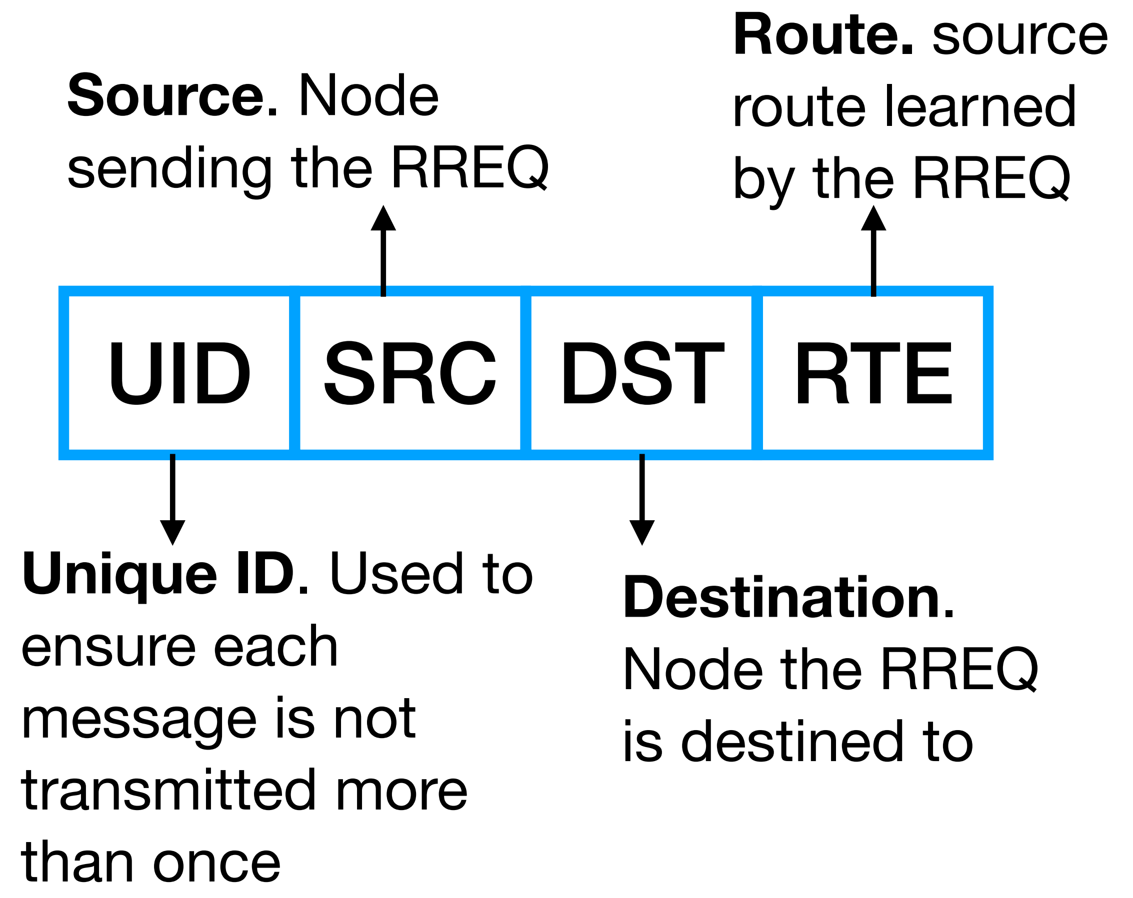
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



The process is repeated until destination is reached

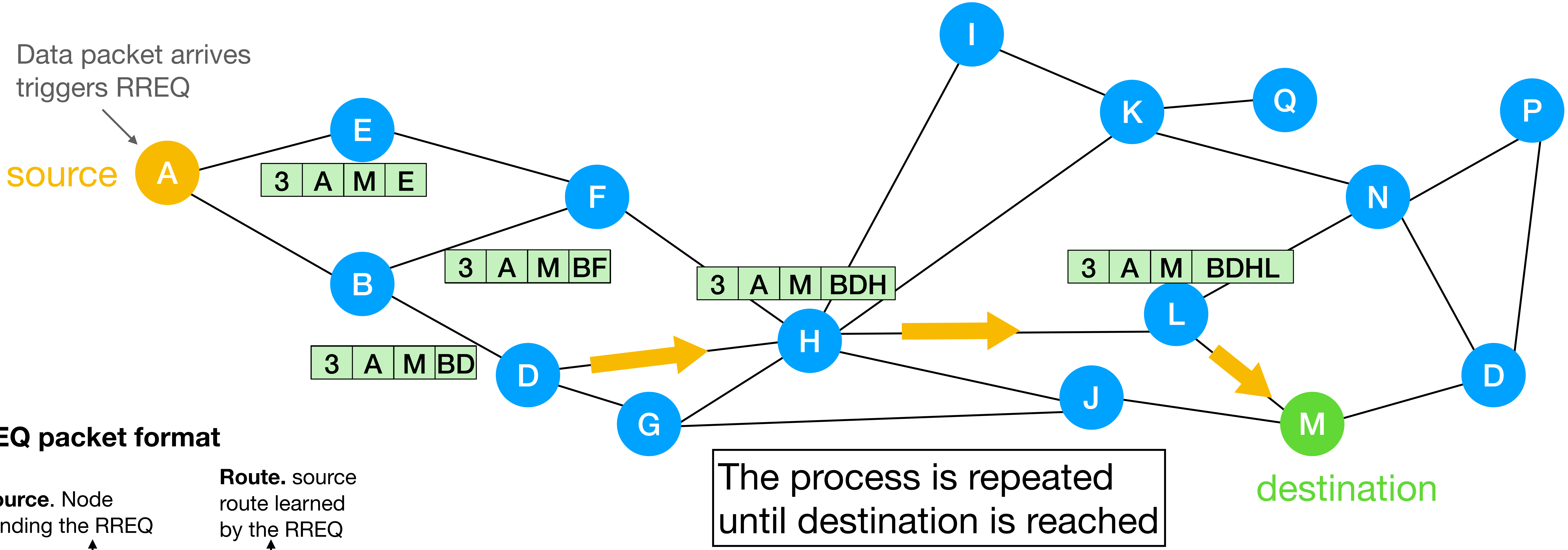
RREQ packet format



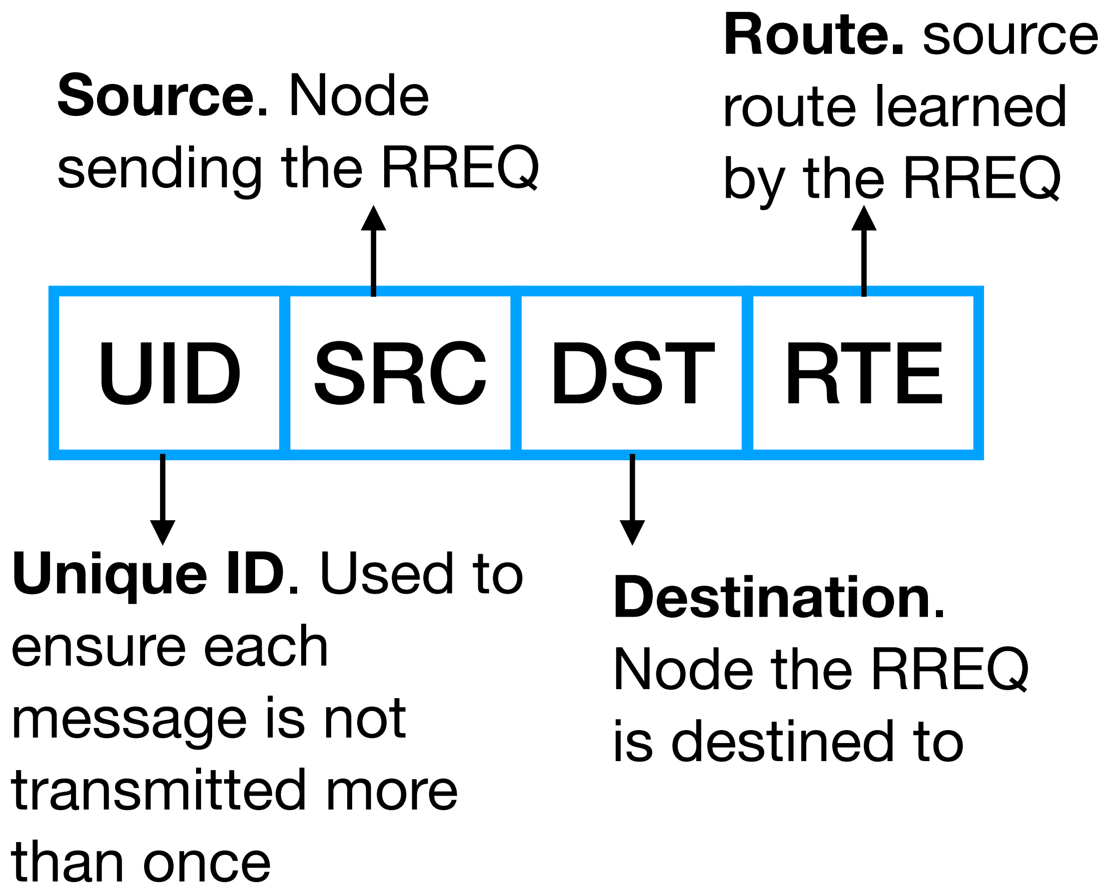
Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.

- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)

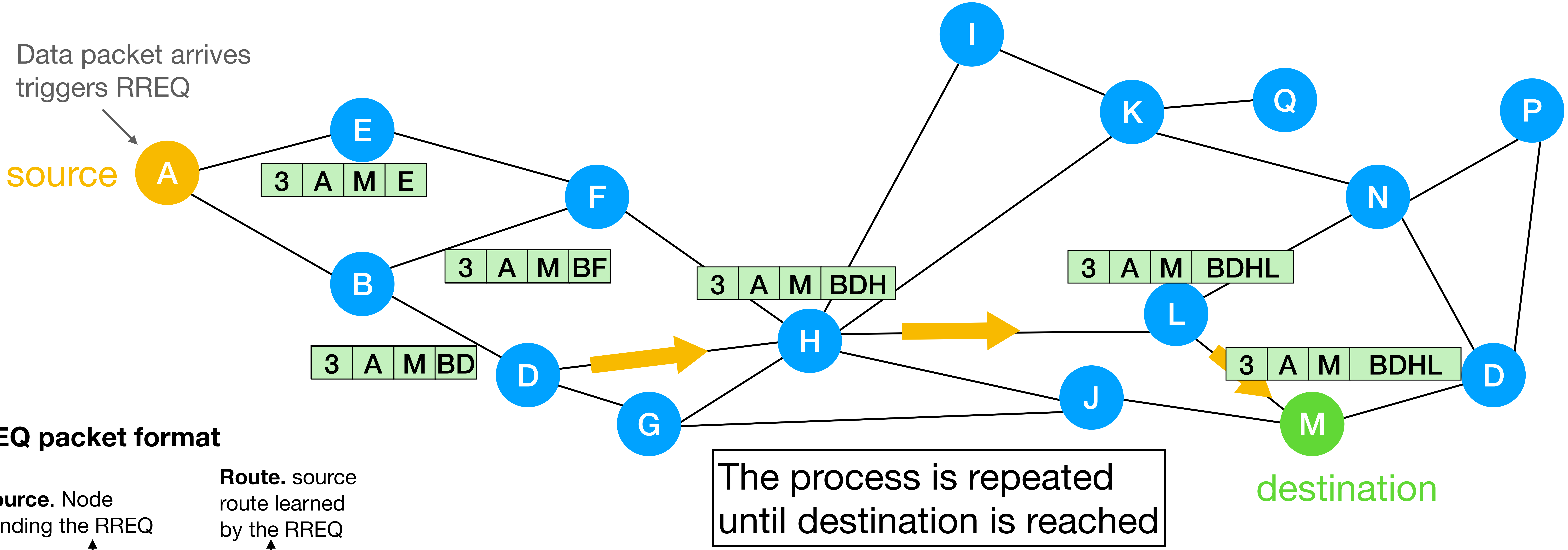


RREQ packet format



Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination. Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREQ (2)



RREQ packet format

UID	SRC	DST	RTE

Source. Node sending the RREQ

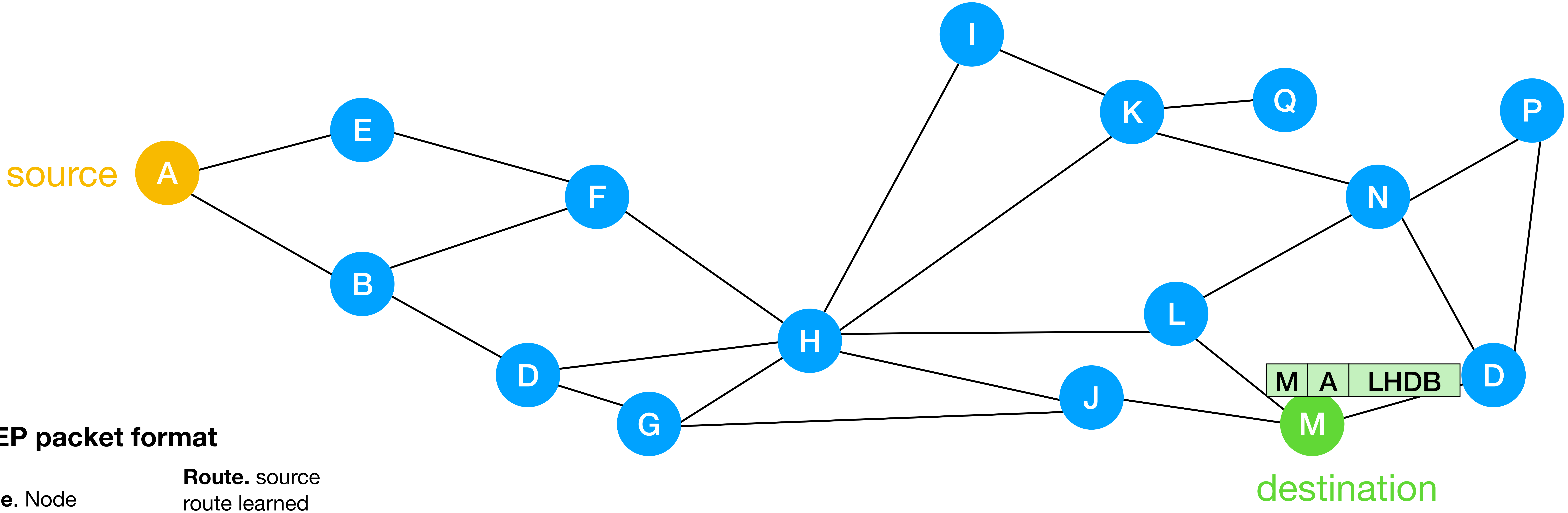
Route. source route learned by the RREQ

Unique ID. Used to ensure each message is not transmitted more than once

Destination. Node the RREQ is destined to

Step 1: on data packet arrival, source broadcasts RREQs to discover path to destination.
Nodes attach their ID in RTE and continue the broadcasting process.
- A node that has already received packet with UID, discards new packet with same UID.
- Might not find the shortest path, but finds a working path source-destination

2 Dynamic Source Routing- RREP



RREP packet format

Source. Node sending the RREP

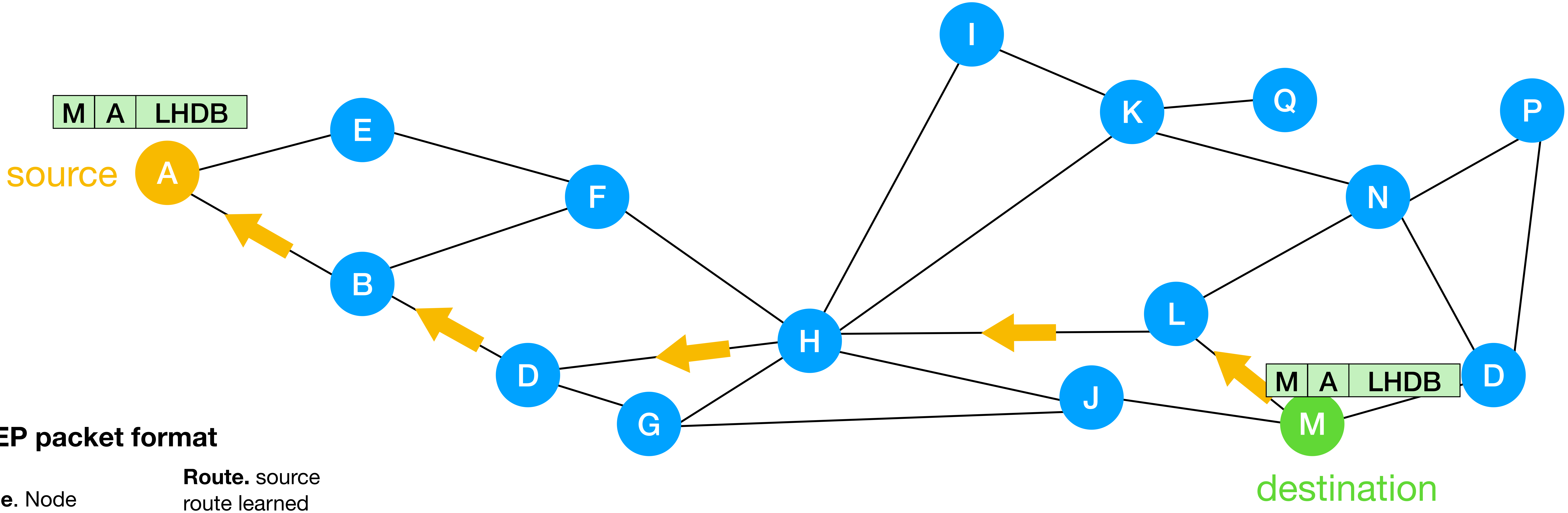
Route. source route learned by the RREQ

SRC	DST	RTE

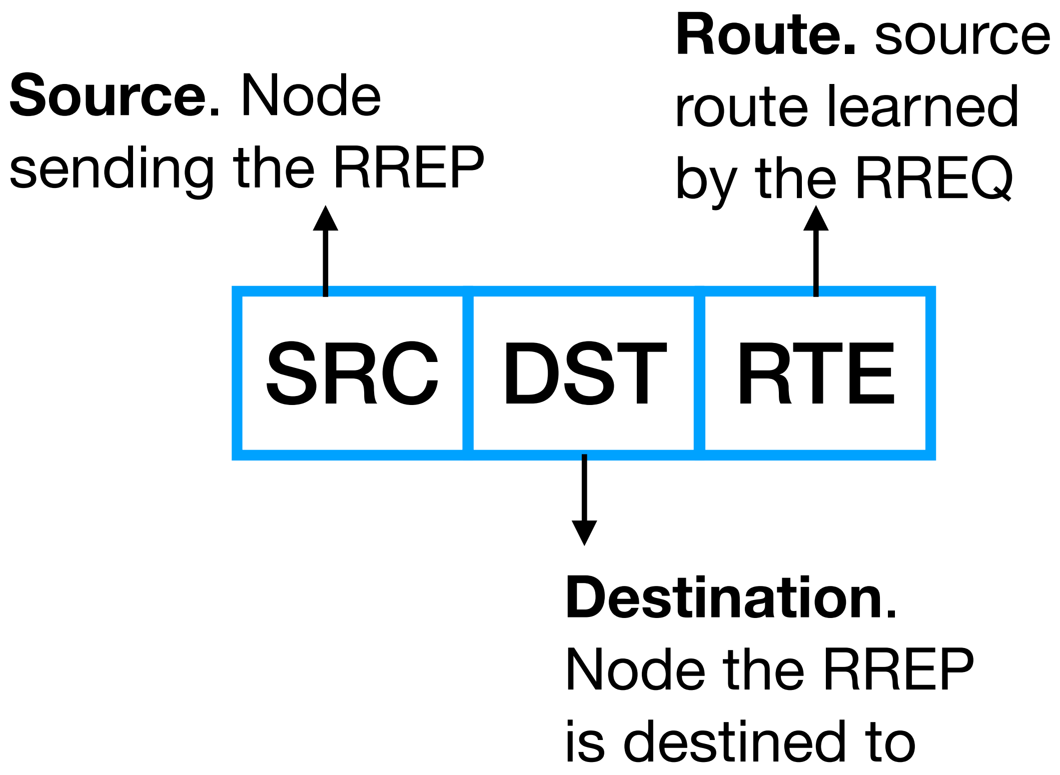
Destination. Node the RREP is destined to

Step 2: destination node replies with RREP, which is not flooded but rather unicast for efficiency along the reverse path contained in the received RREQ

2 Dynamic Source Routing- RREP

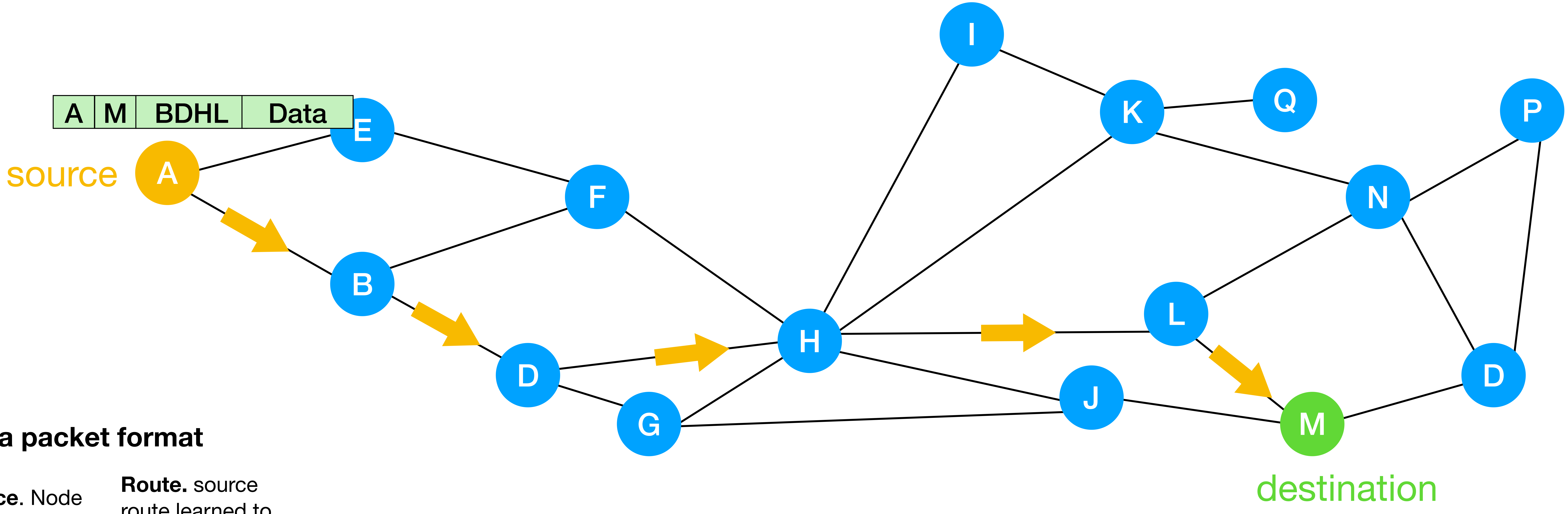


RREP packet format

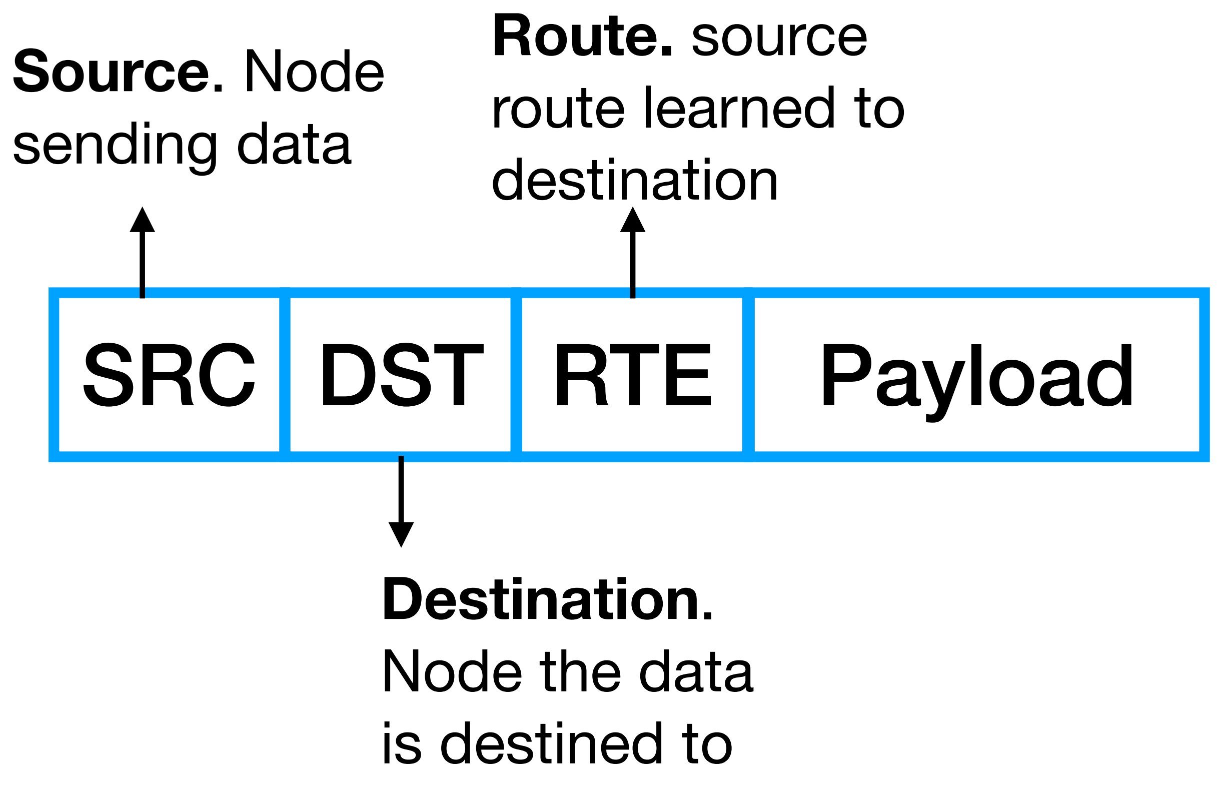


Step 2: destination node replies with RREP, which is not flooded but rather unicast for efficiency along the reverse path contained in the received RREQ

2 Dynamic Source Routing- forwarding



Data packet format



Step 3: Upon receiving RREP, source can send data through the RTE. Intermediate node simply read the RTE to forward packet to the next hop in the path

2 Dynamic Source Routing (DSR) - Route caching

- To improve efficiency, we can implement **caching**, i.e., source will cache route for some period of time, in case it wants to send more packets to that same destination later

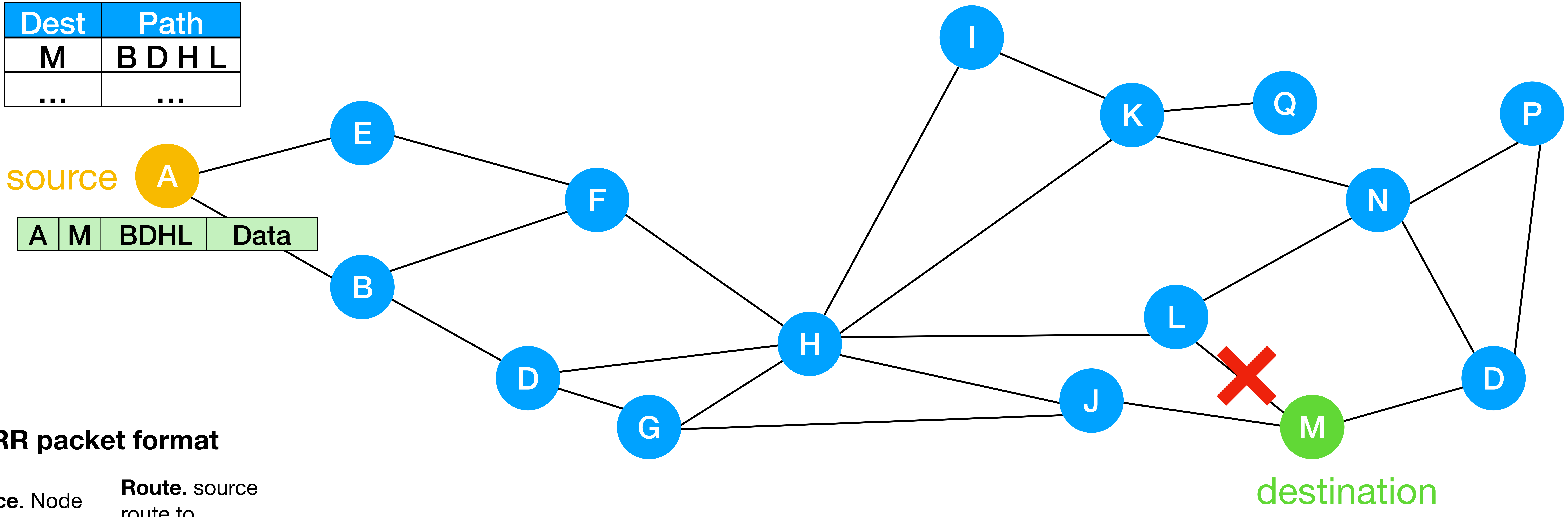
- Intermediate nodes and other nodes overhearing RREQs and RREPs may also cache they see

- Entries are deleted after timeout (tunable parameter).
  - highly dynamic network -> set a low timeout,
  - mostly static network -> set a high timeout

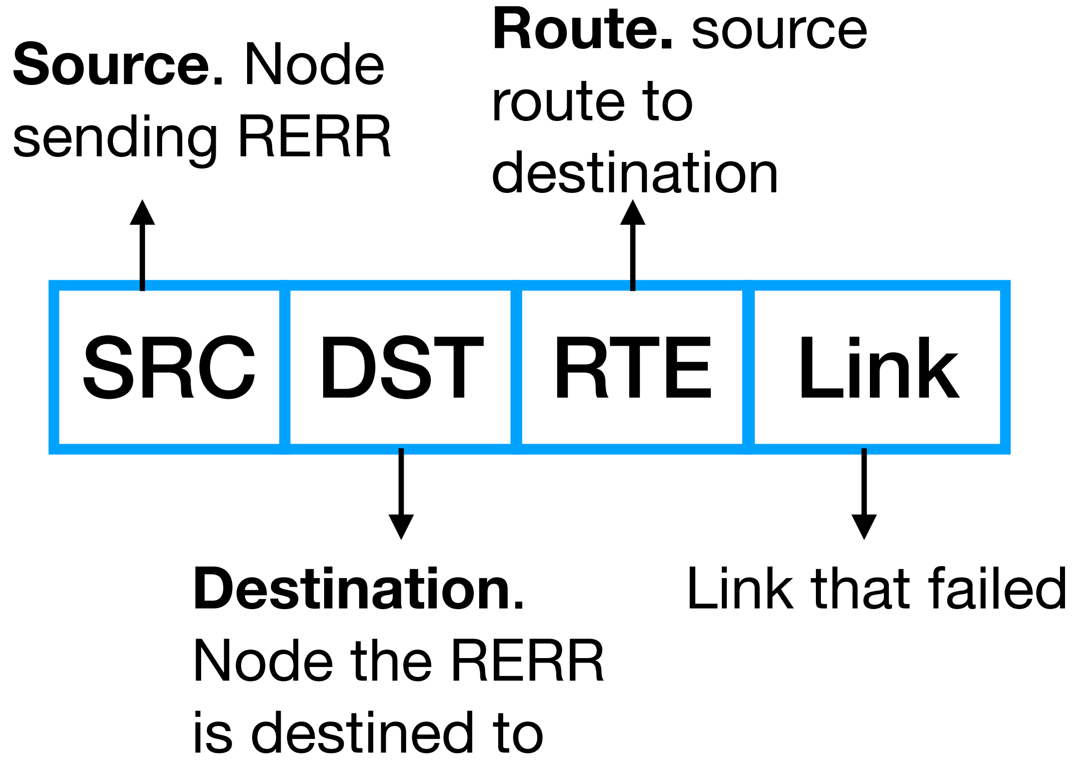
- Route cache size can be limited (tunable parameter).

2 Dynamic Source Routing- Route Error (RERR)

Dest	Path
M	B D H L
...	...



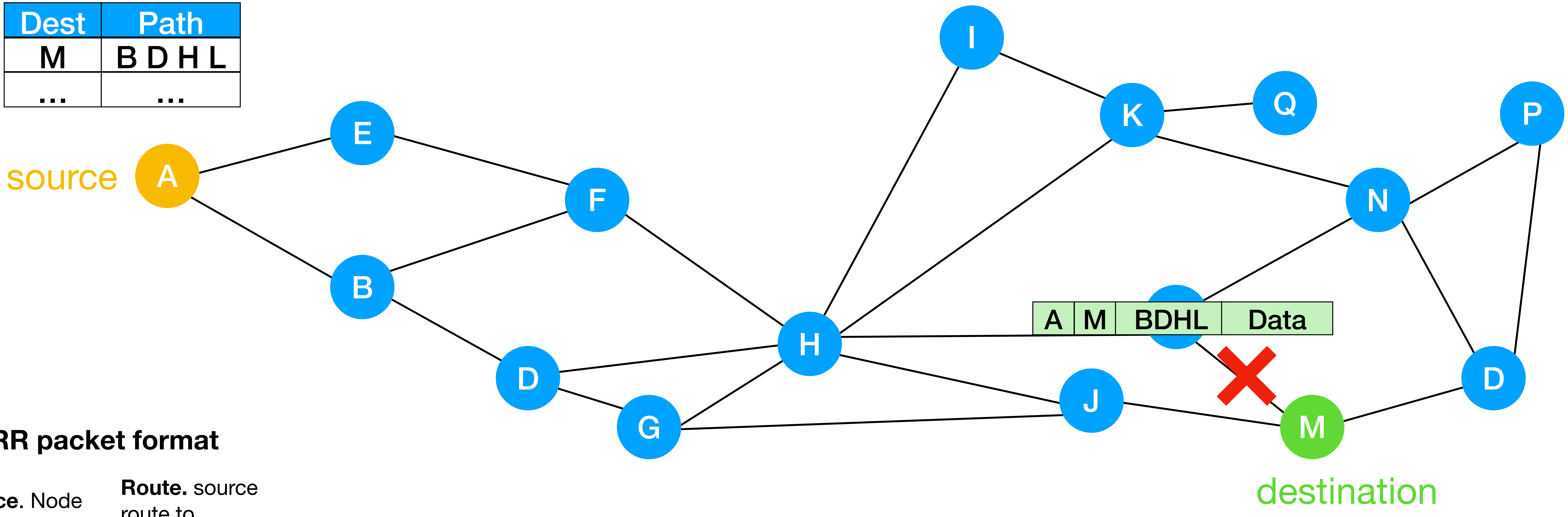
RERR packet format



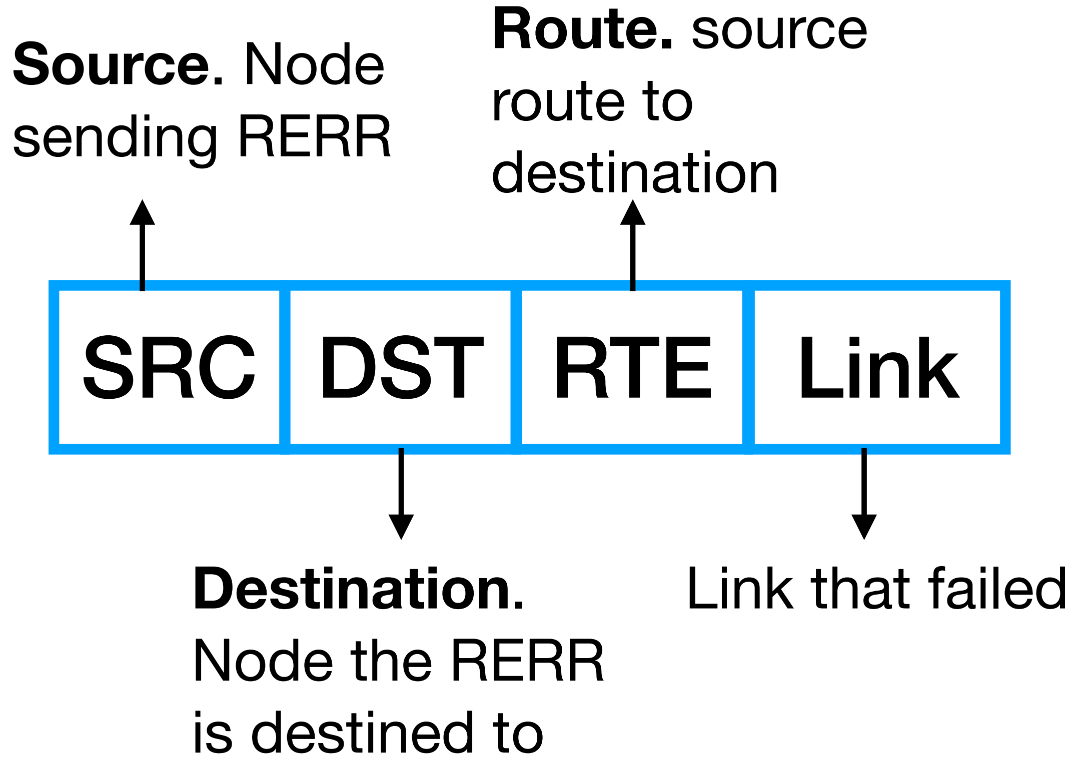
Route Error Messages are sent **reactively** when a failure is detected.

2 Dynamic Source Routing- Route Error (RERR)

Dest	Path
M	B D H L
...	...



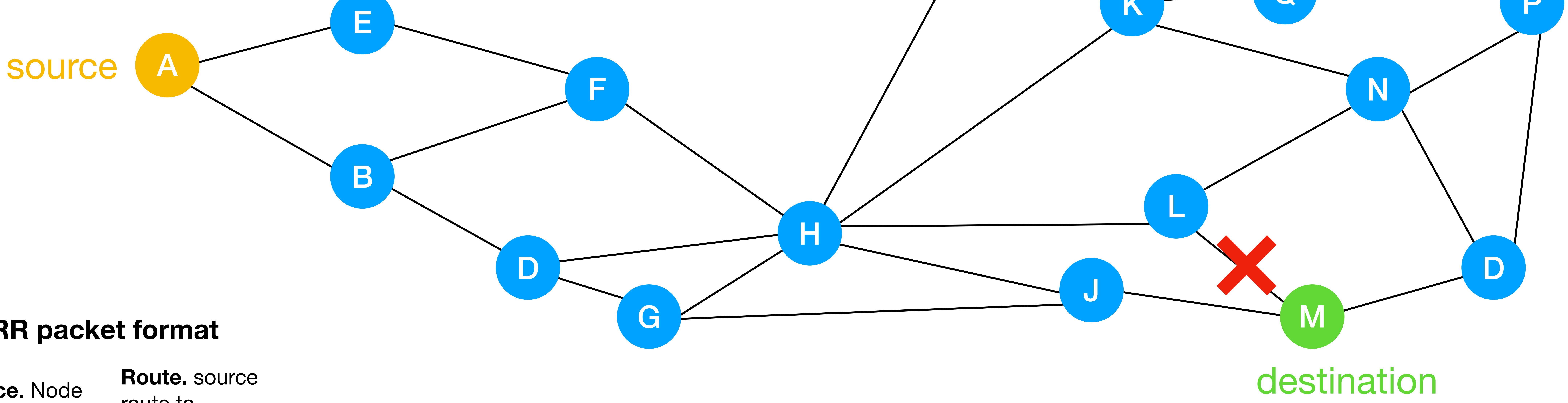
RERR packet format



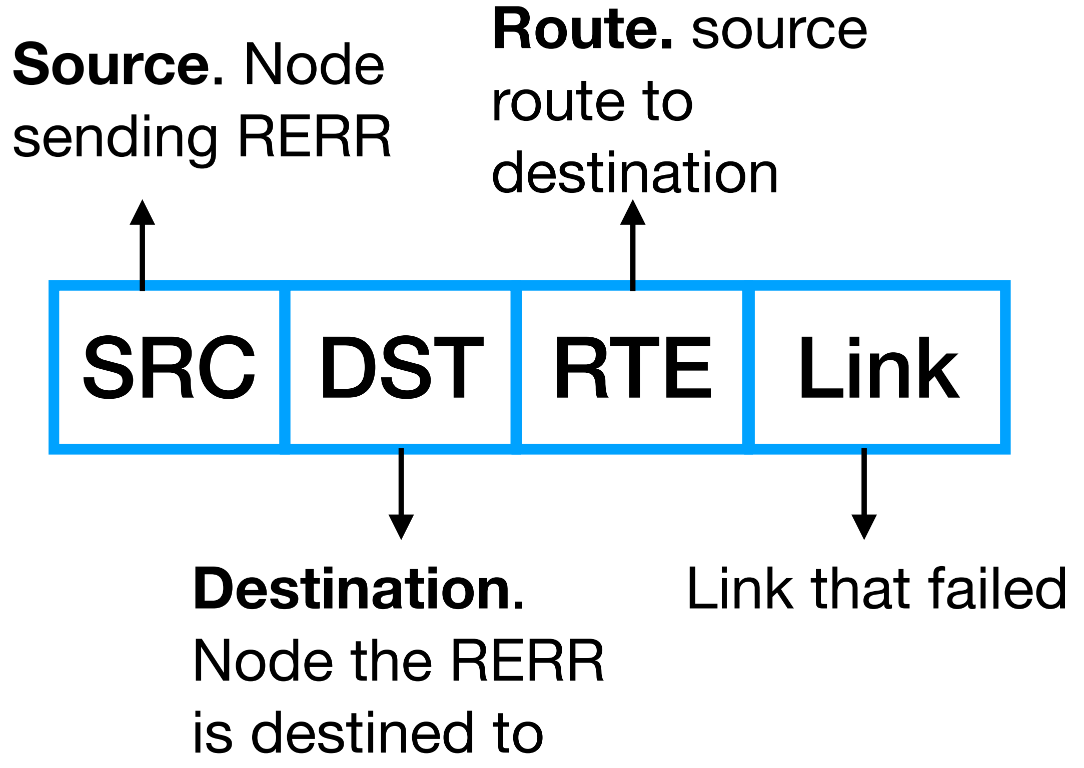
Route Error Messages are sent **reactively** when a failure is detected.

Dynamic Source Routing- Route Error (RERR)

Dest	Path
M	B D H L
...	...



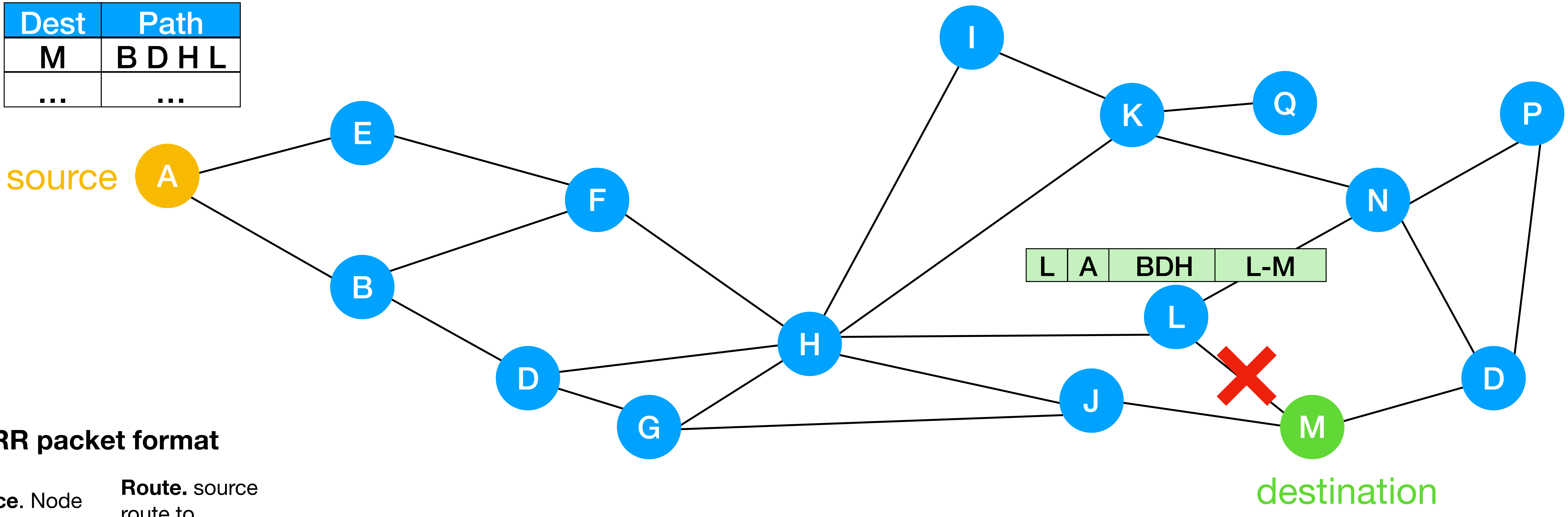
RERR packet format



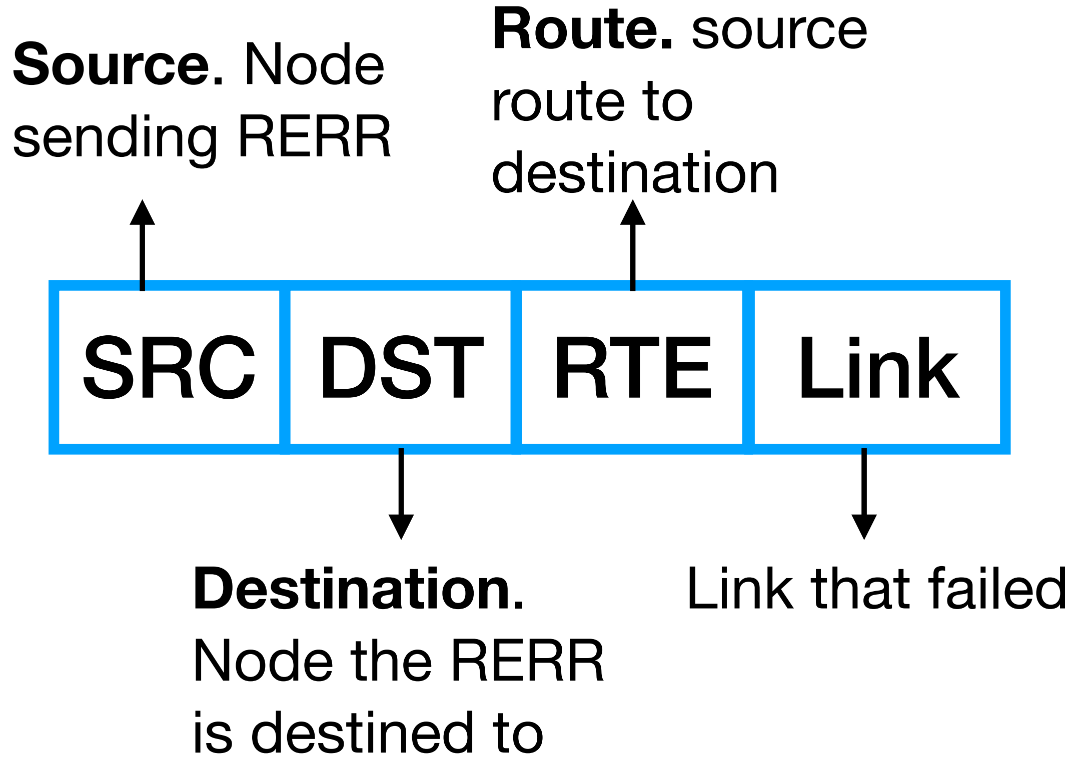
Route Error Messages are sent **reactively** when a failure is detected.

2 Dynamic Source Routing- Route Error (RERR)

Dest	Path
M	B D H L
...	...

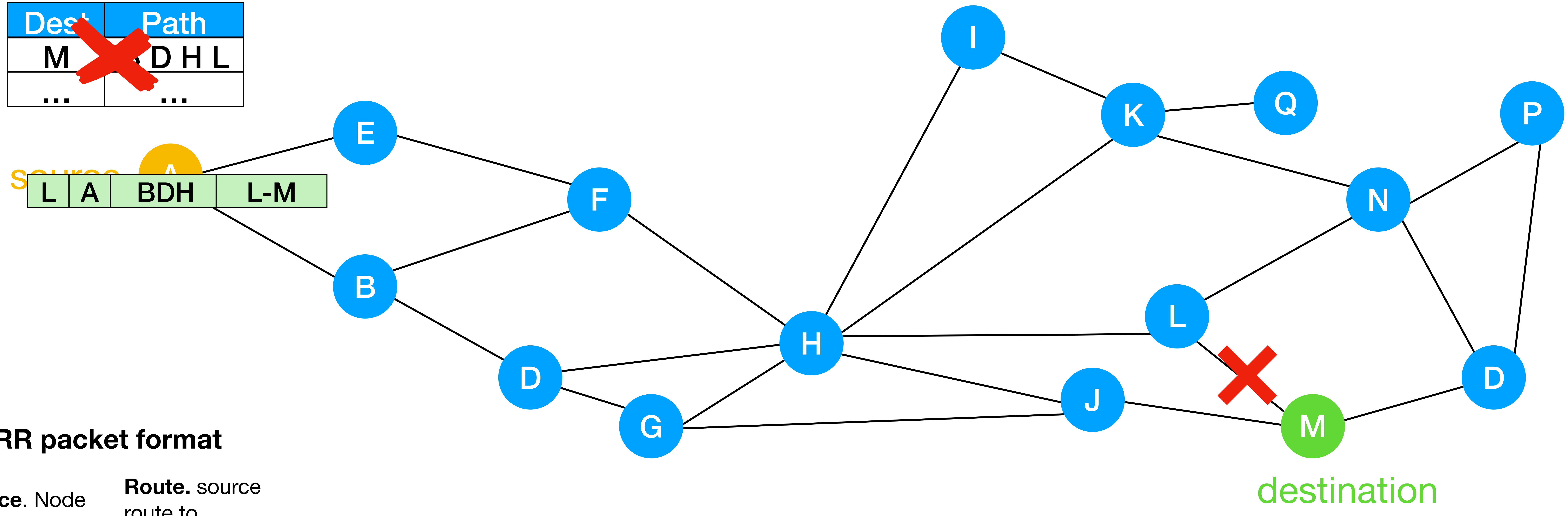


RERR packet format

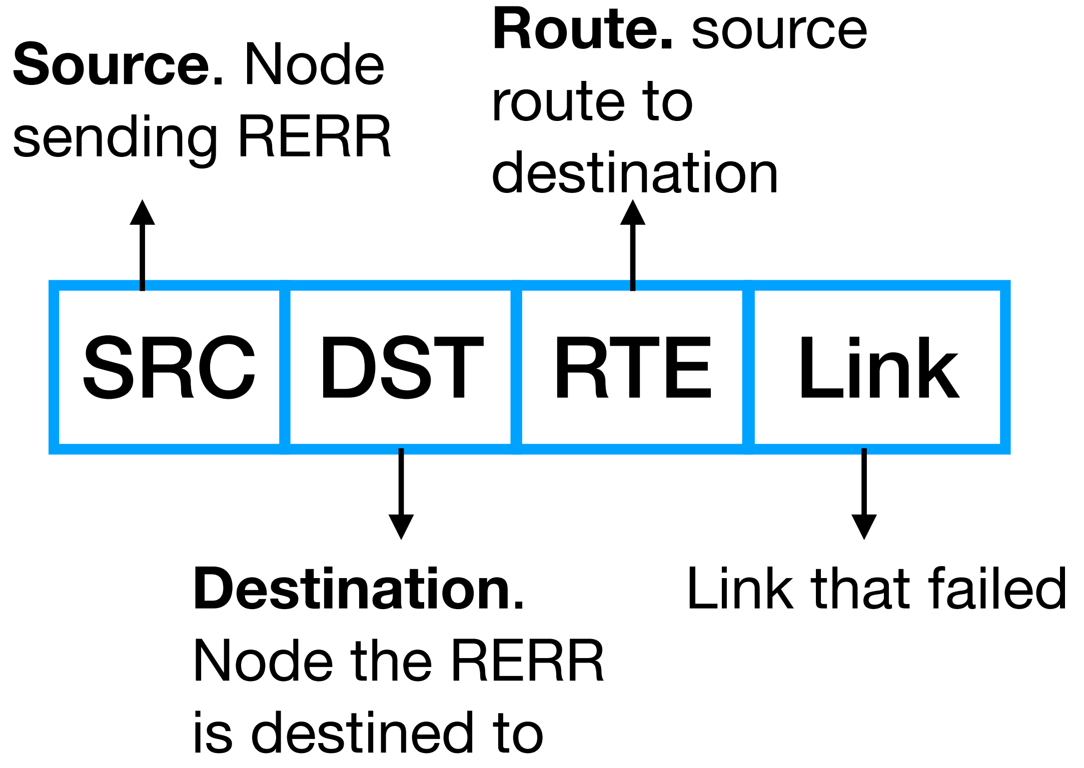


Route Error Messages are sent **reactively** when a failure is detected.

Dynamic Source Routing- Route Error (RERR)



RERR packet format



Route Error Messages are sent **reactively** when a failure is detected.

2 Dynamic Source Routing (DSR) (2)

- The idea behind source routing has been around for decades. In fact, the Internet technically supports source routing (there are fields in the IPv4 specification that allow source routing of packets). But it is deprecated. Can be good for IoT applications.
- Downsides:
  - as the network gets big, source routes get long - packet overhead
  - caches can get stale (outages not discovered until packets are sent - reactive routing is “lazy”)
- Idea: distribute information more in the network (more like link state, embed the path in the network rather than on the packets)

Full documentation: https://datatracker.ietf.org/doc/html/rfc4728

3 Ad Hoc On-demand Distance Vector (AODV) (1)

- Reactive routing algorithm (no route discovery process until data has to be sent) with Route Request messages (RREQ) to discover a route.
- RREQ creates distance-vector entries at each hop pointing back to source.
- Nodes on active path maintain routing information.
- Convergence issues typical in distance Vector can be restrained by introducing a destination-controlled **sequence number** stored with each route.
  - The destination increments the value of the sequence number every time there is an event (like a failure or a new node coming up).

Similarly to DSR

Unlike DSR

3

Ad Hoc On-demand Distance Vector - Route Request (RREQ)

Originator Sequence number. Current sequence number of the source

Broadcast ID. Unique identifier of the RREQ. Ensures each message is not transmitted more than once

Source. ID Node sending the RREQ

RREQ packet format

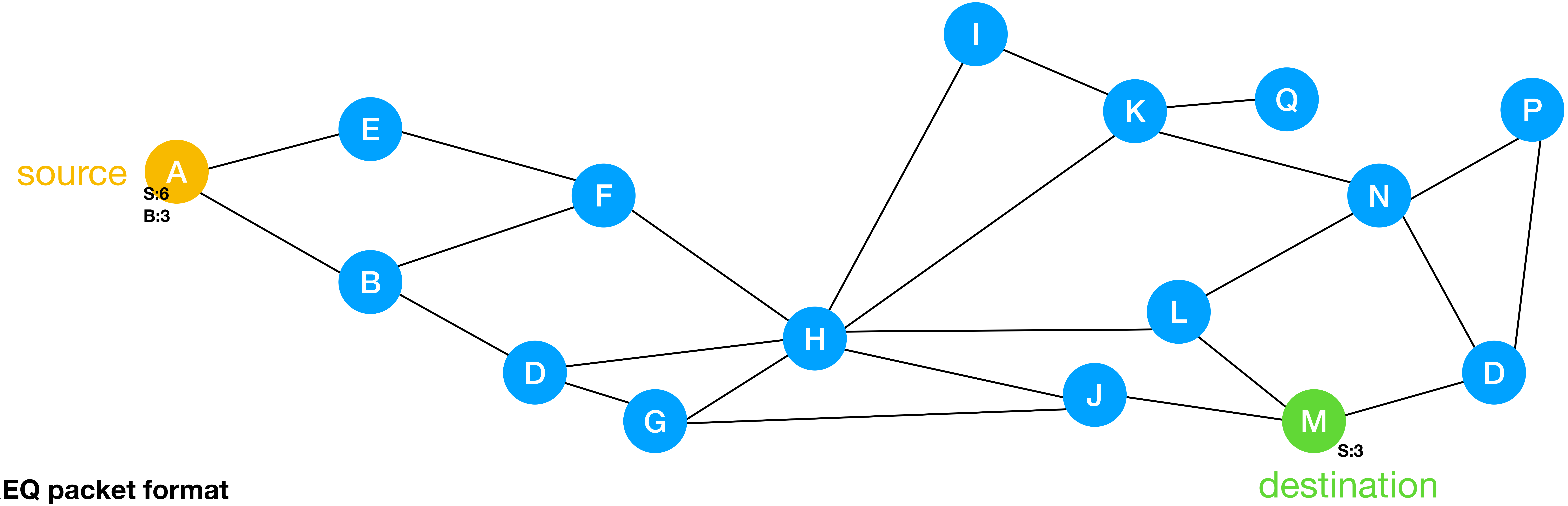
HOPS	RREQ ID	DST	DST SEQ N	SRC	SRC SEQ N

Hop Count. Number of transmissions needed to reach destination. Each node receiving RREQ increment HOPS by 1 if it is not the destination

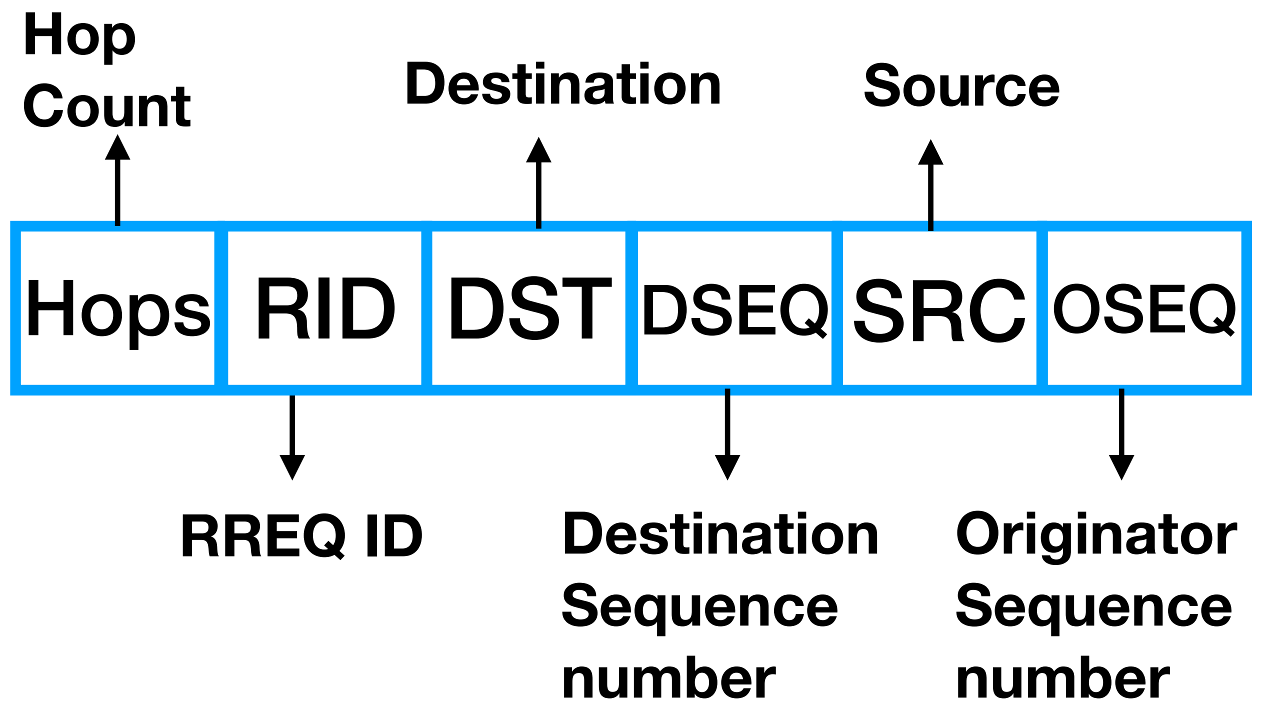
Destination. ID node the RREQ is destined to

Destination Sequence number. The latest sequence number received in the past by the originator for any route towards the destination. Could be unknown (set a flag).

3 Ad Hoc On-demand Distance Vector - RREQ



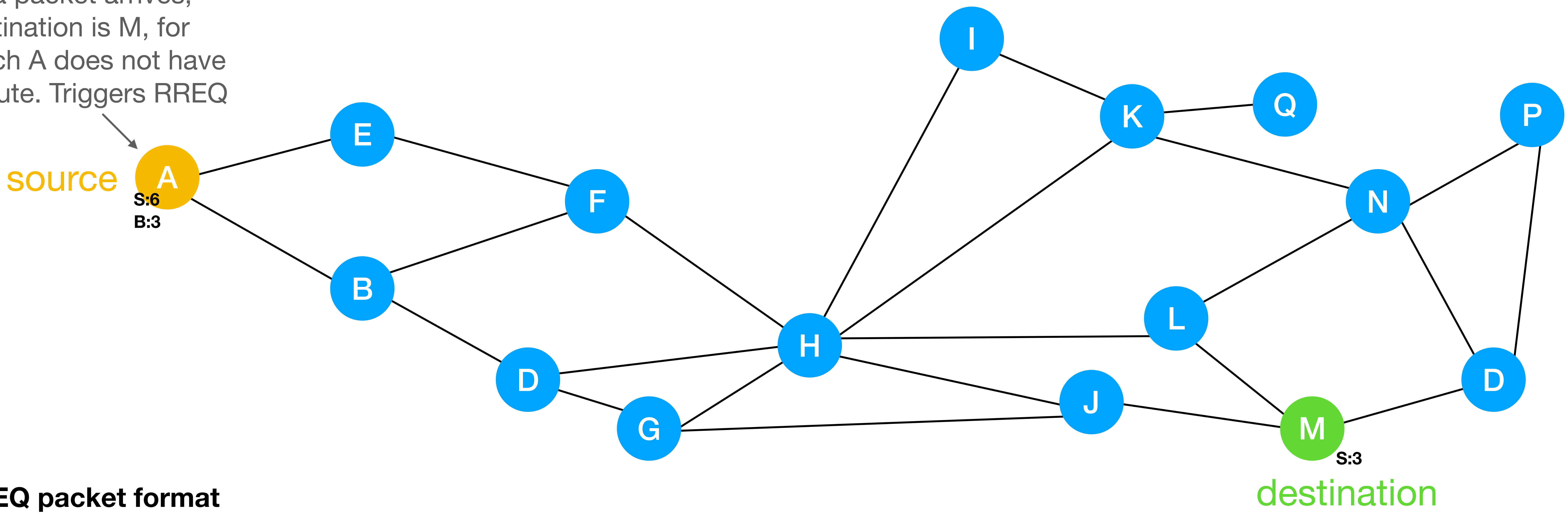
RREQ packet format



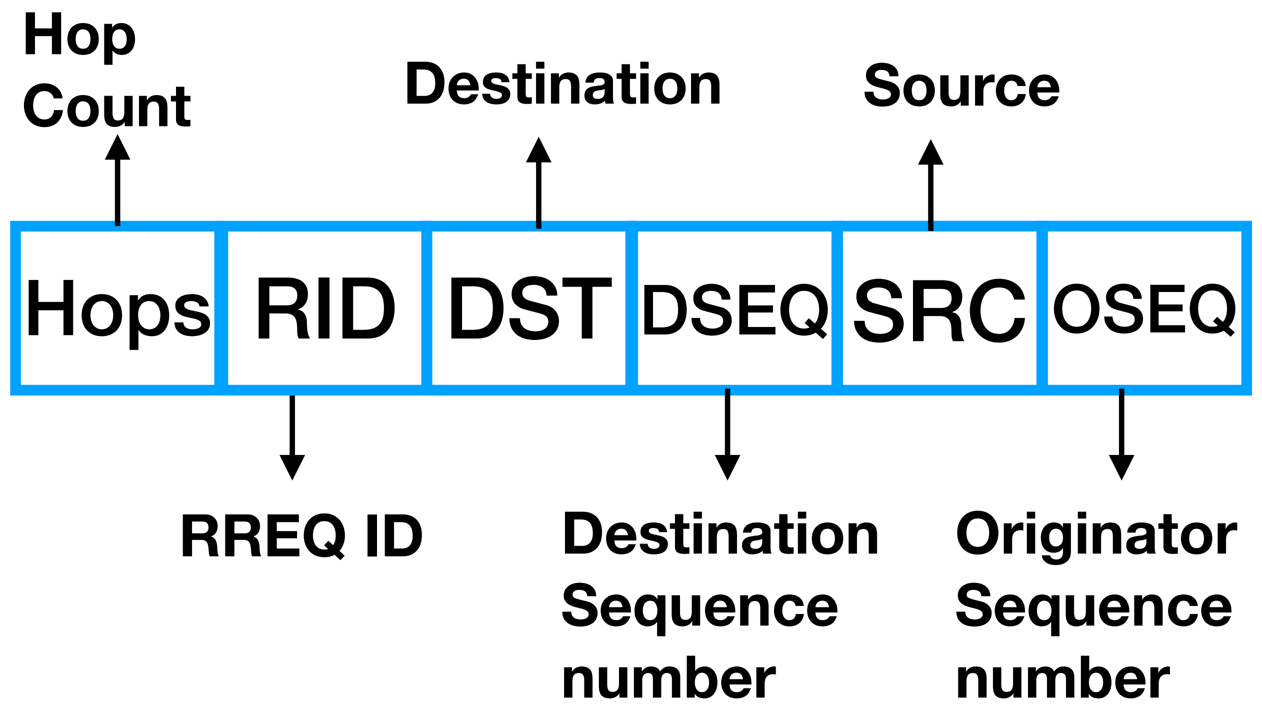
Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ



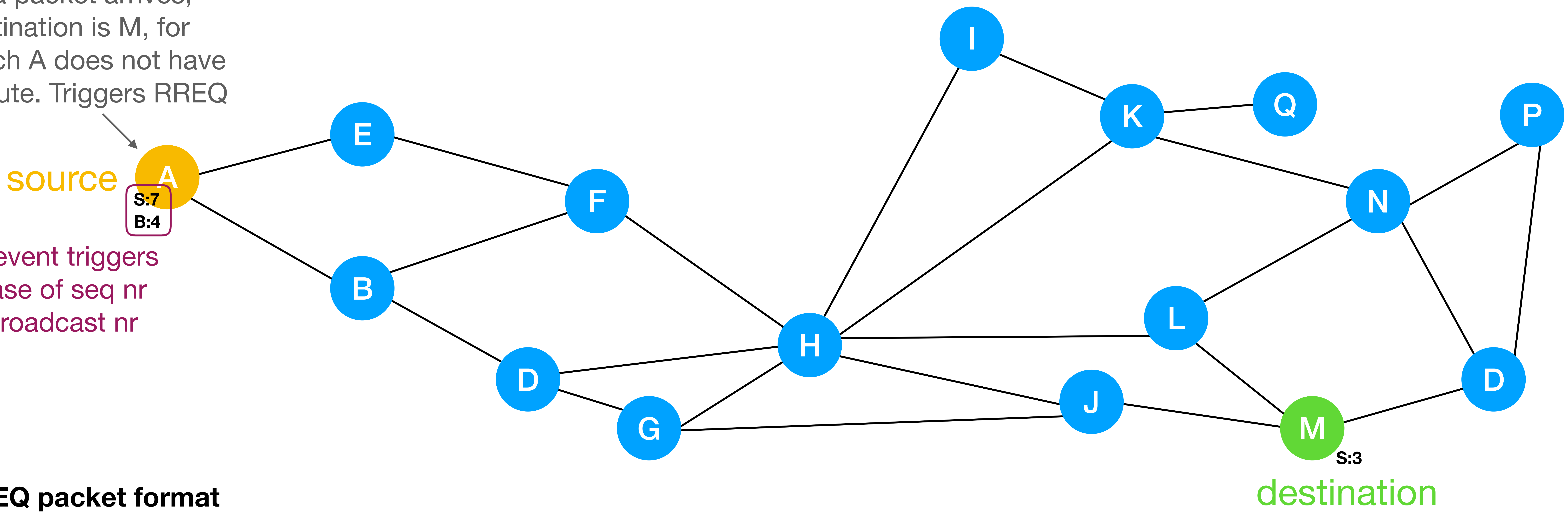
RREQ packet format



Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

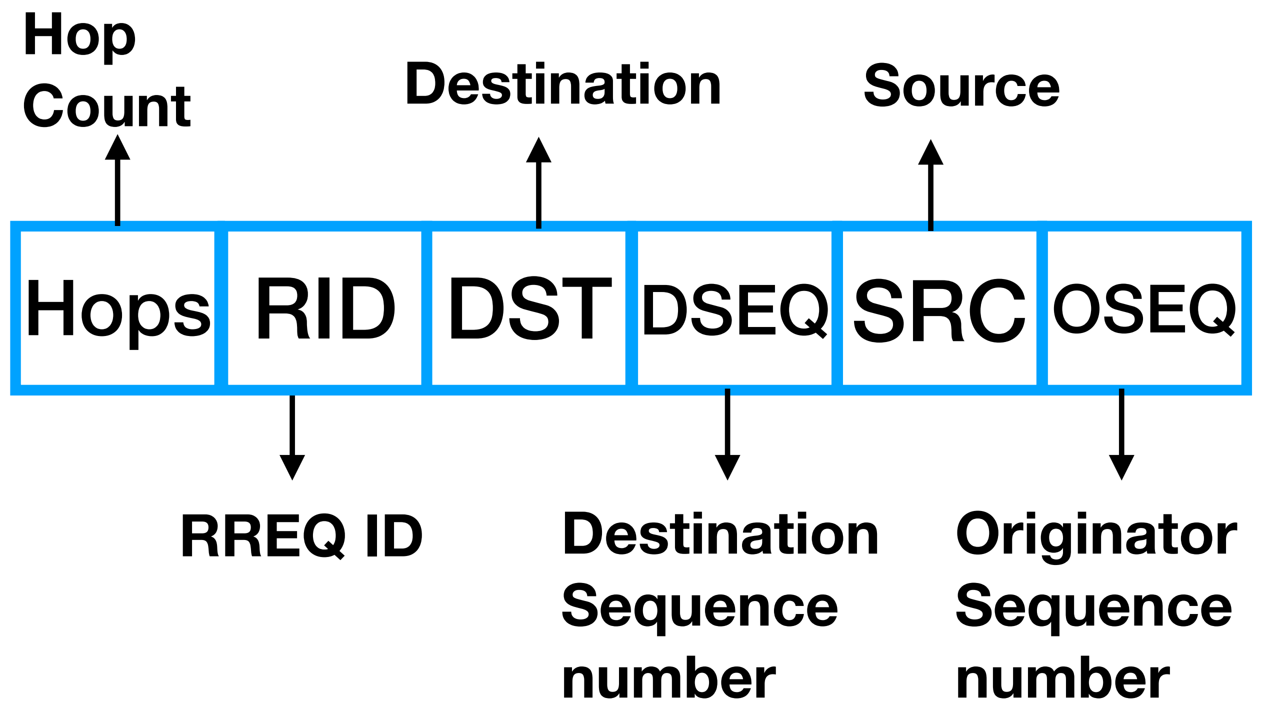
3 Ad Hoc On-demand Distance Vector - RREQ

Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ



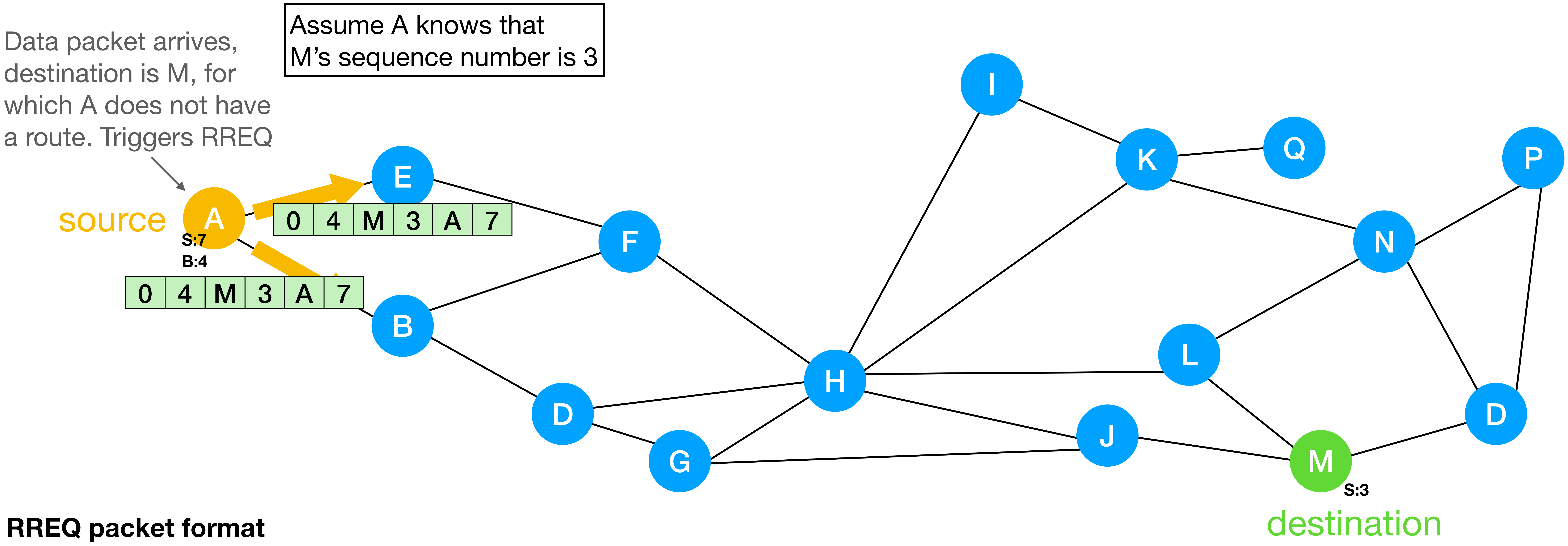
New event triggers increase of seq nr and broadcast nr

RREQ packet format

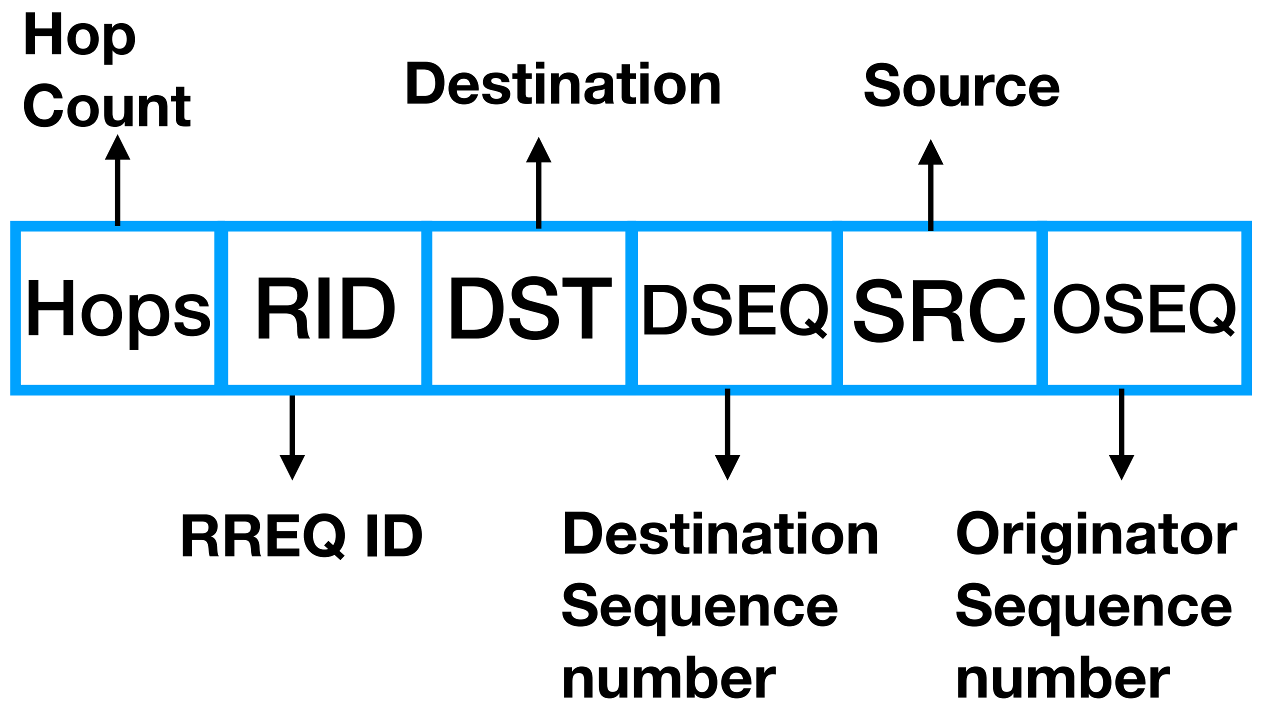


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ



RREQ packet format

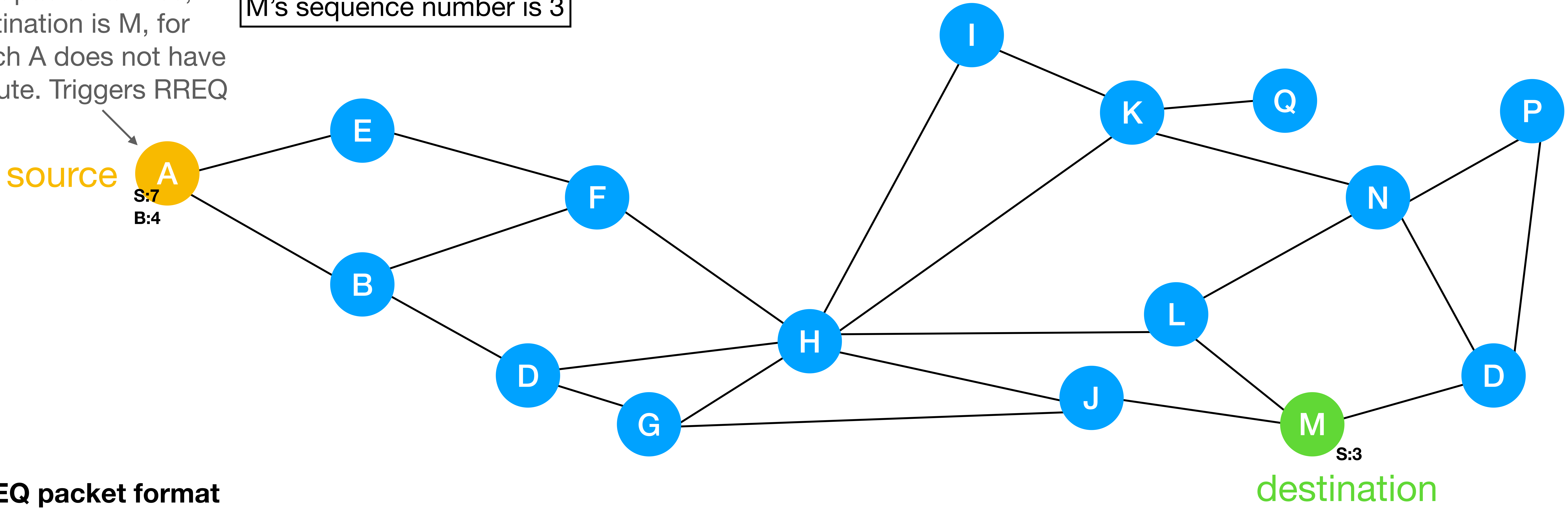


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ

Assume A knows that M's sequence number is 3



RREQ packet format

Hops	RID	DST	DSEQ	SRC	OSEQ
Hop Count	RREQ ID	Destination	Destination Sequence number	Source	Originator Sequence number

Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

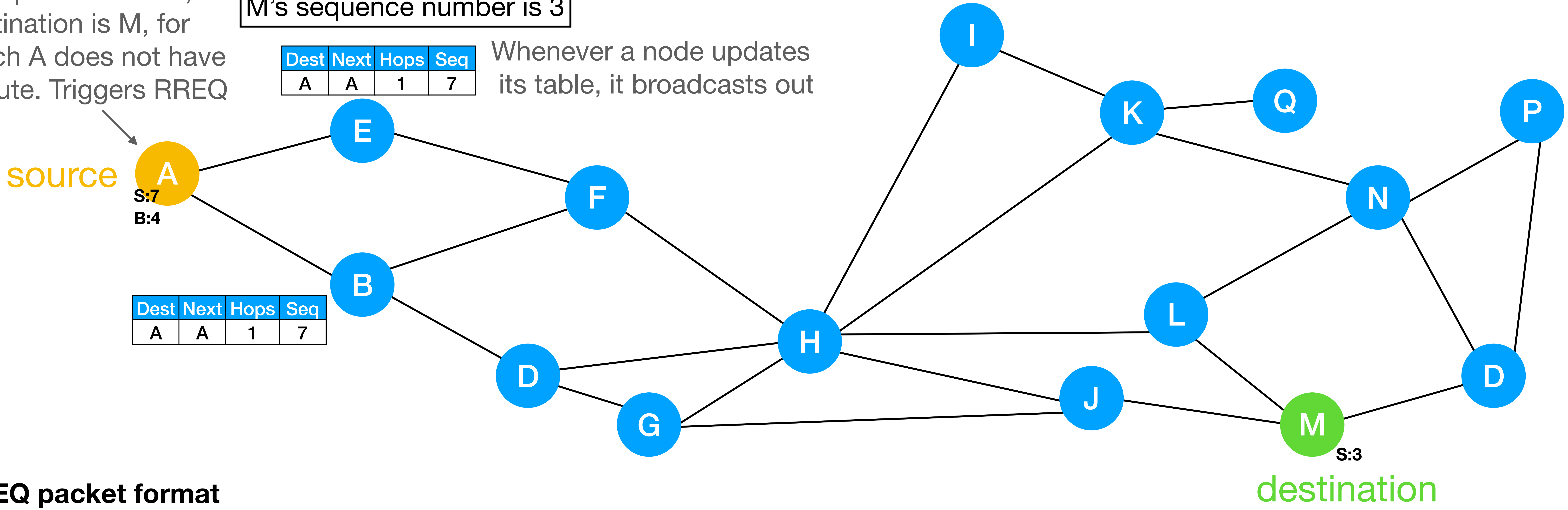
Ad Hoc On-demand Distance Vector - RREQ

3

Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ

Assume A knows that M’s sequence number is 3

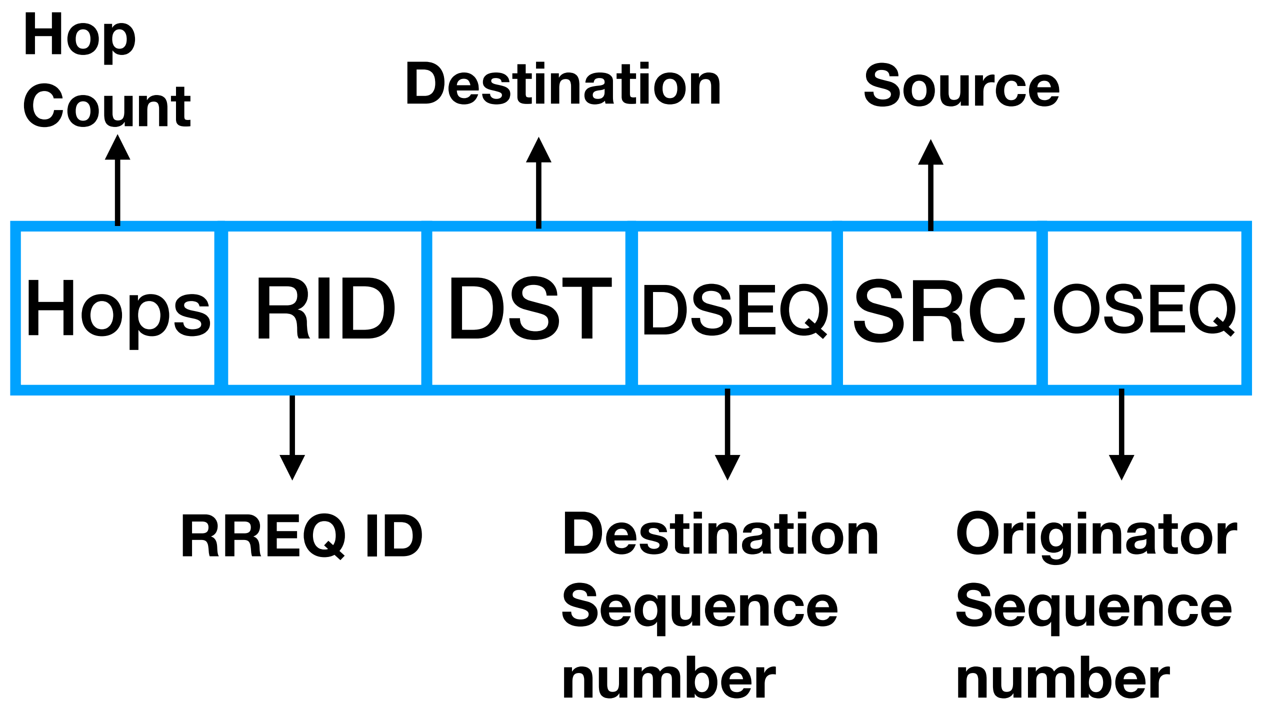
Whenever a node updates its table, it broadcasts out



Dest	Next	Hops	Seq
A	A	1	7

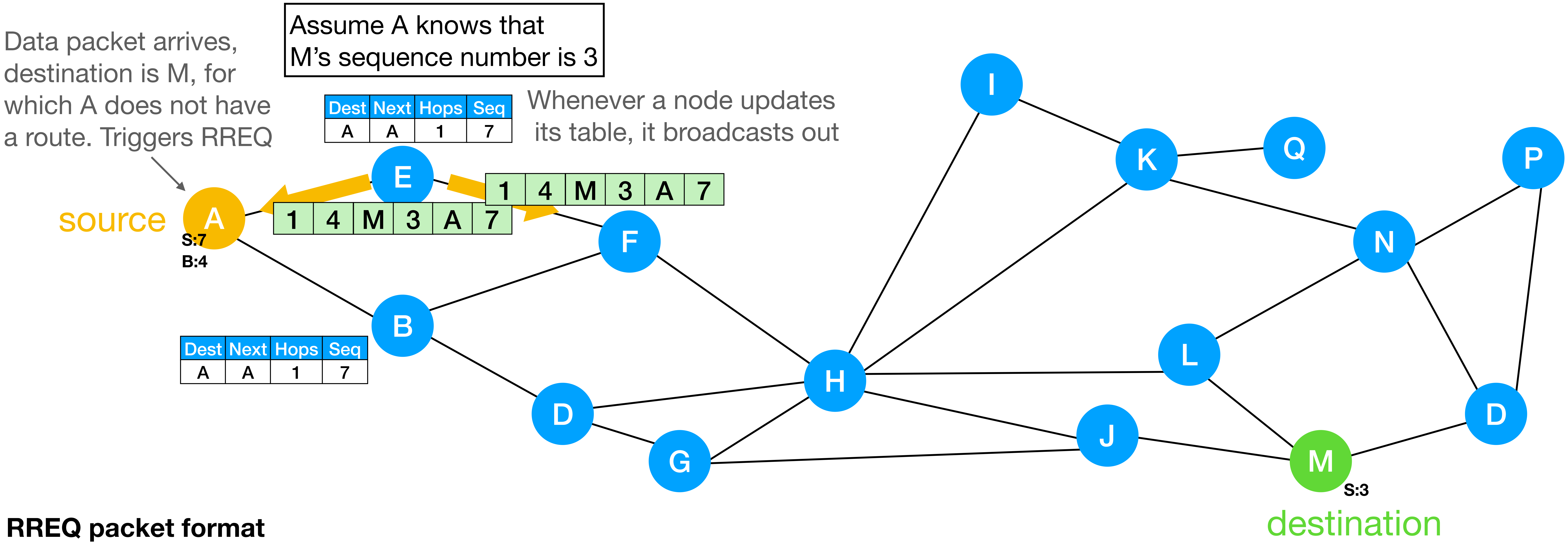
Dest	Next	Hops	Seq
A	A	1	7

RREQ packet format

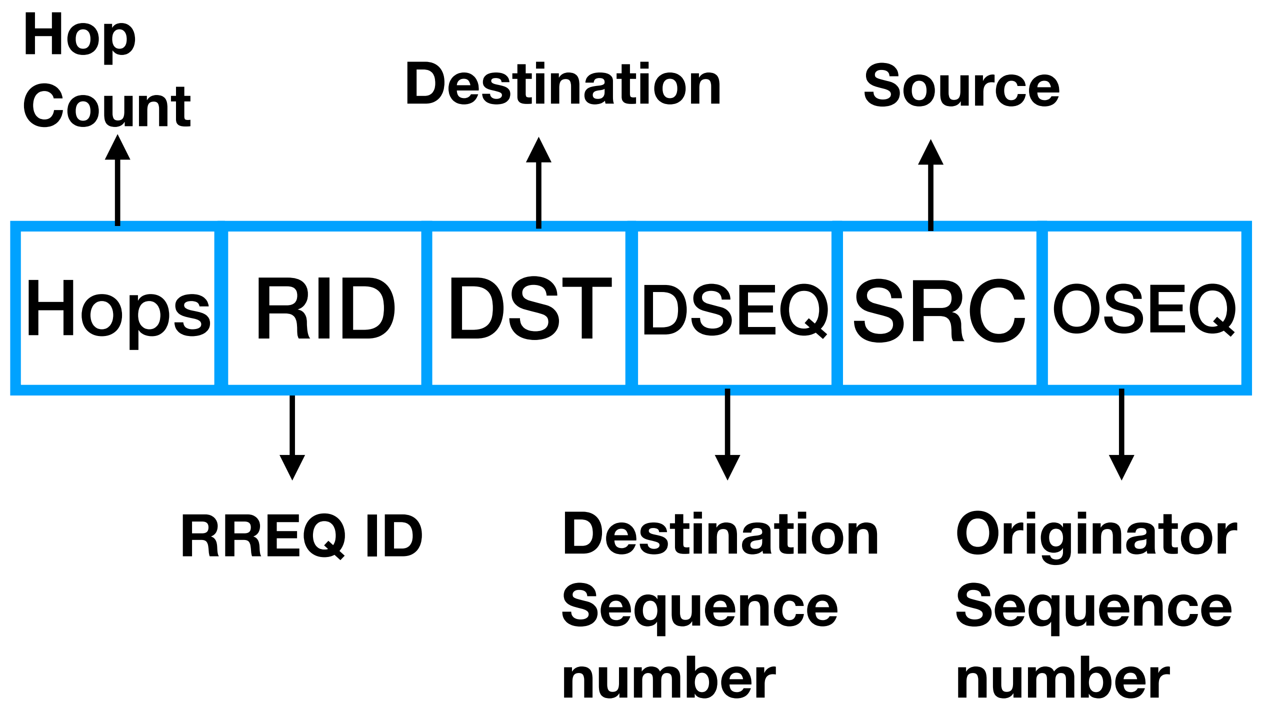


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

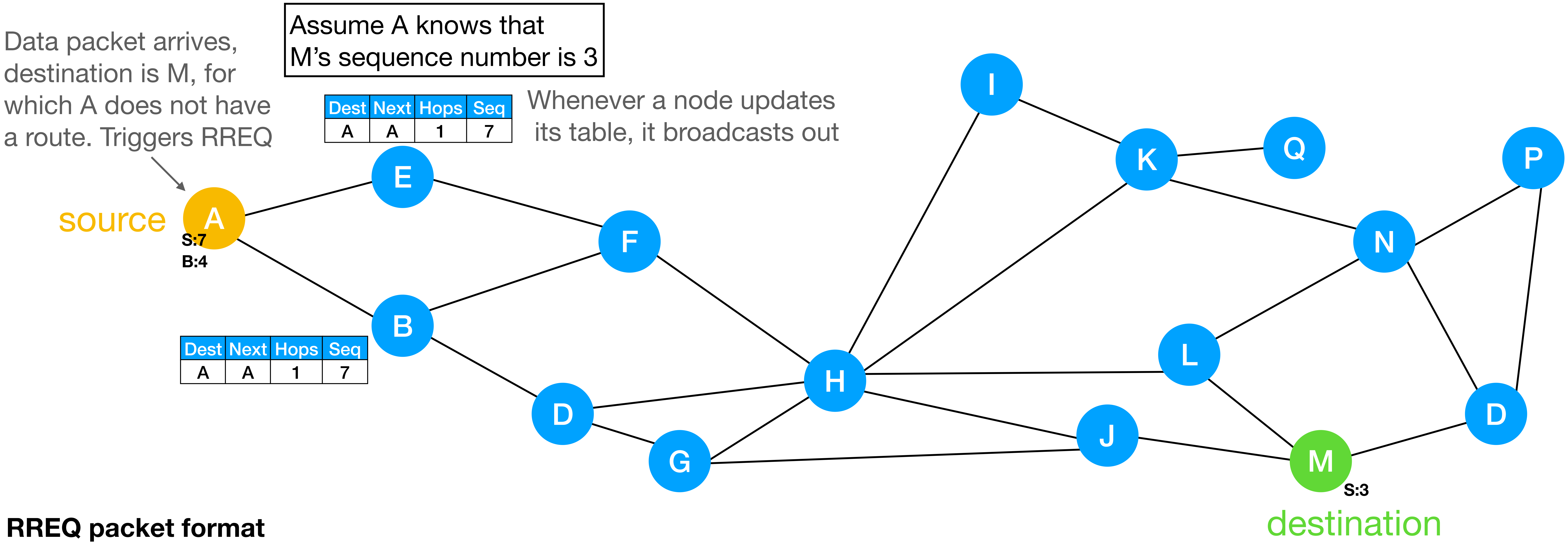


RREQ packet format

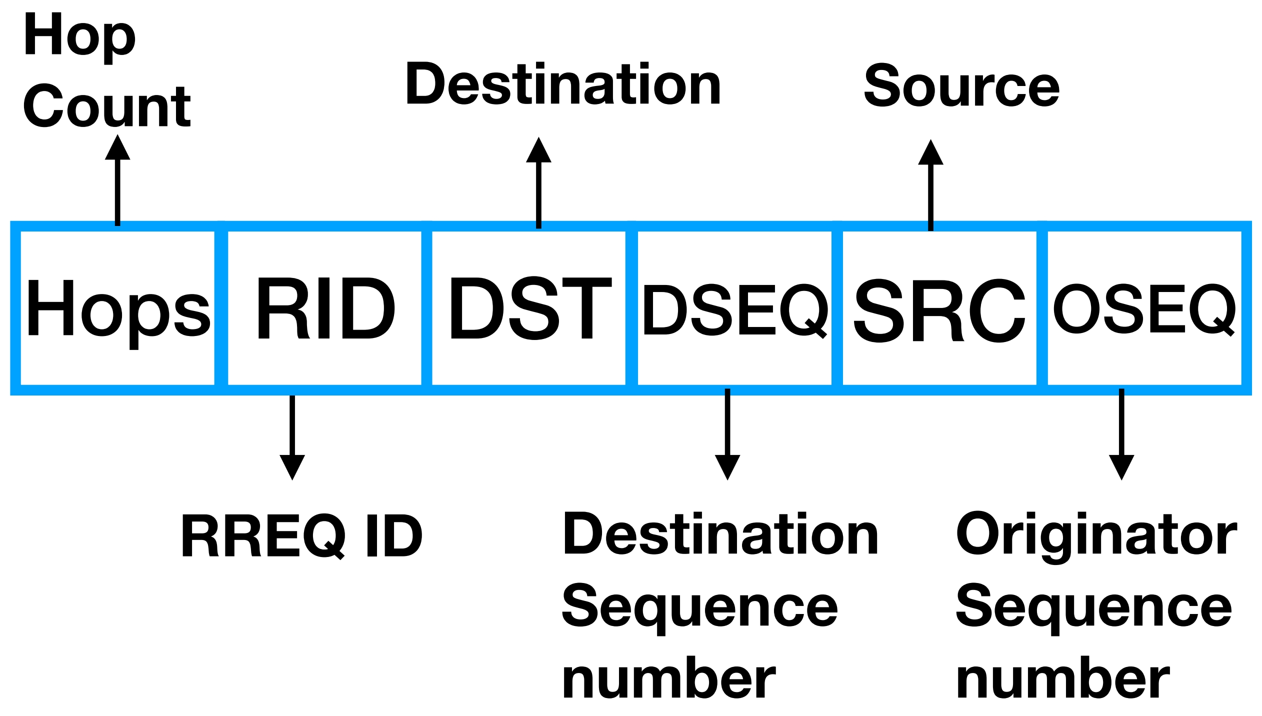


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ



RREQ packet format



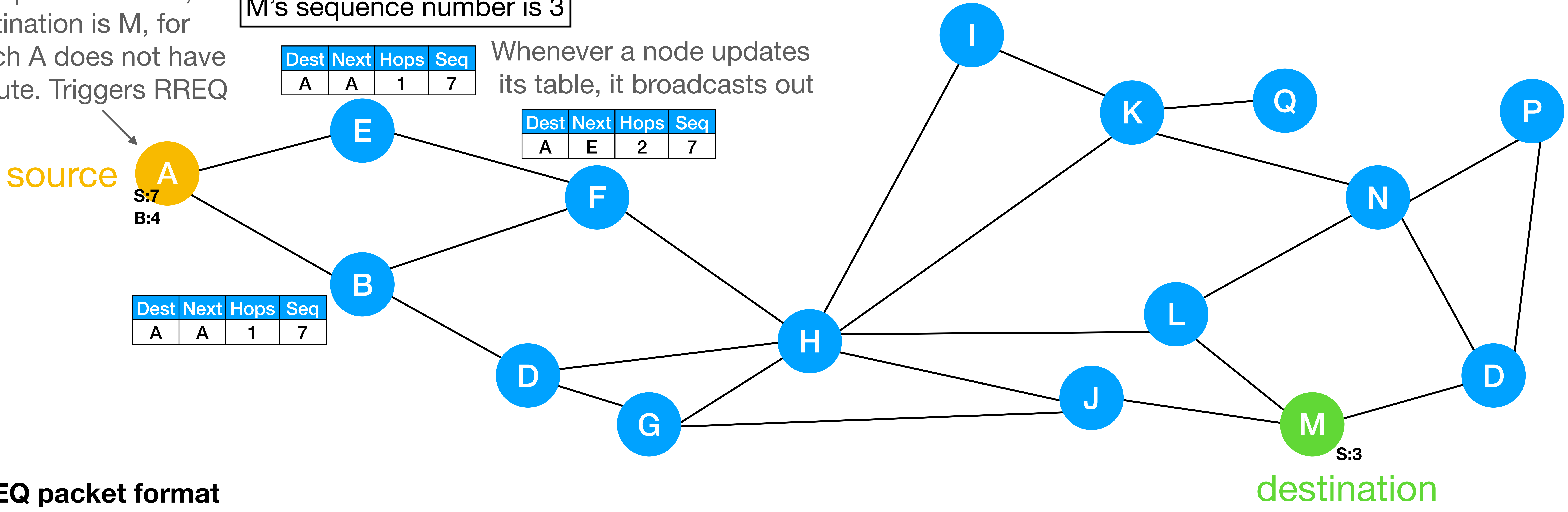
Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

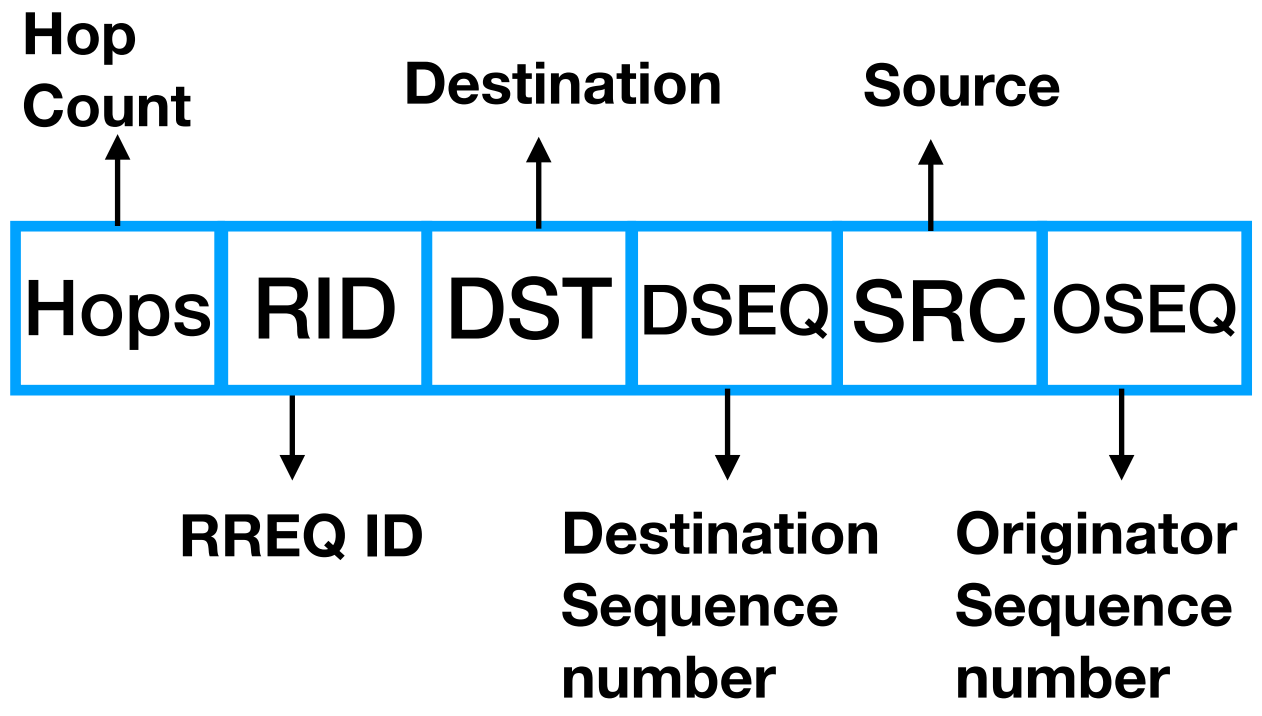
Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ

Assume A knows that M’s sequence number is 3

Whenever a node updates its table, it broadcasts out

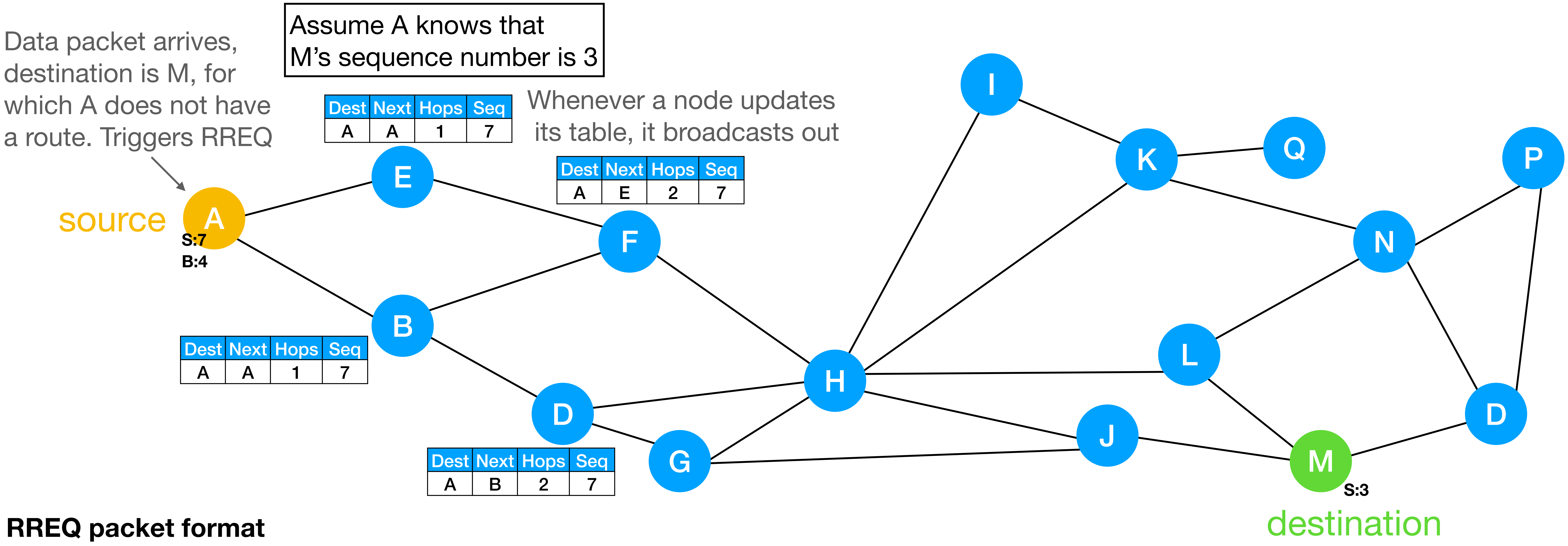


RREQ packet format

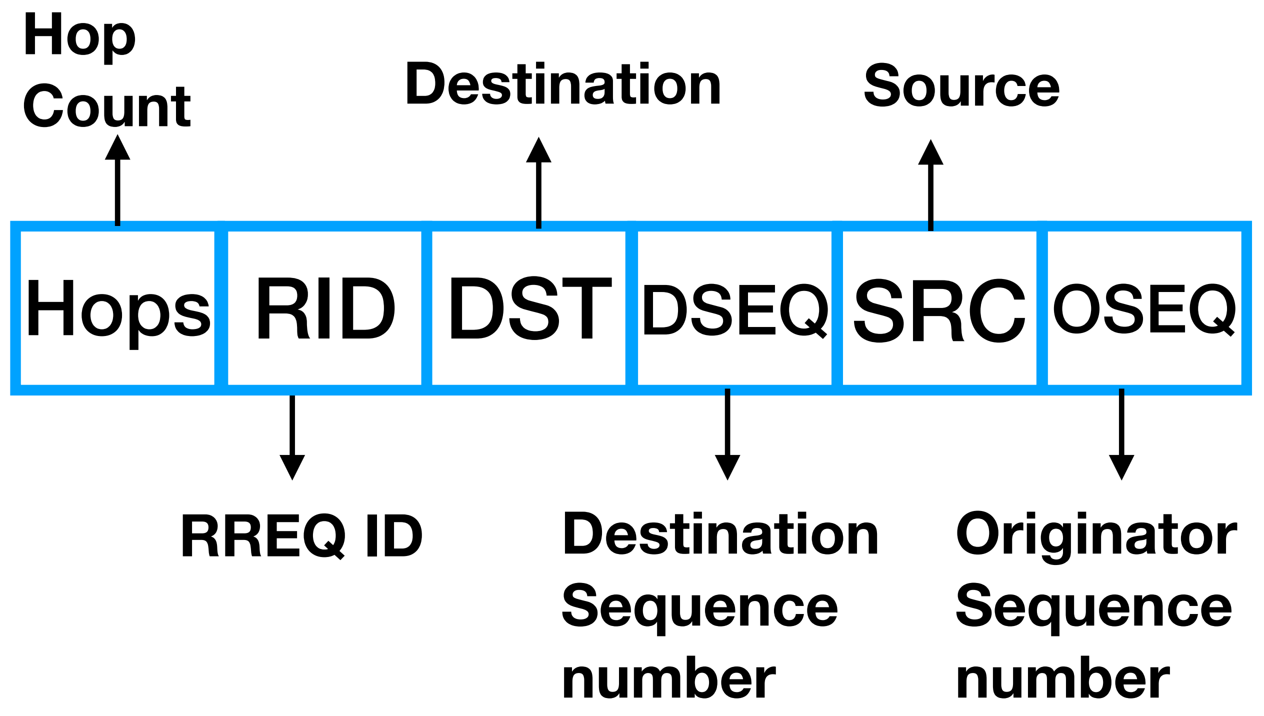


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ



RREQ packet format

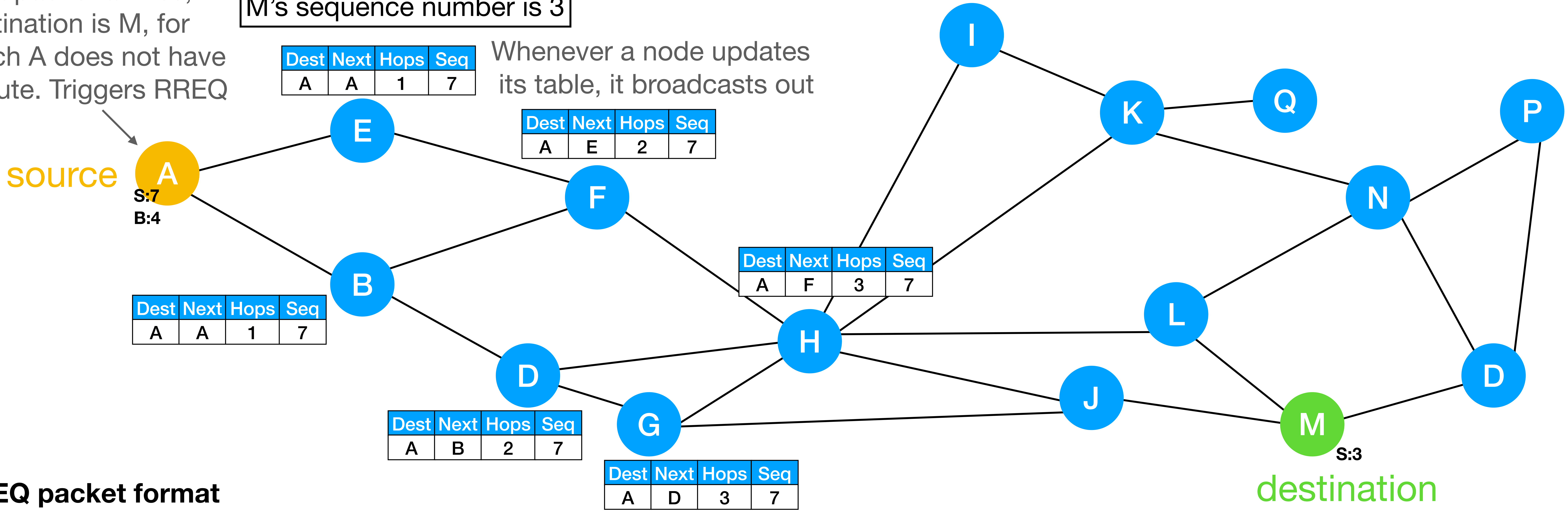


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

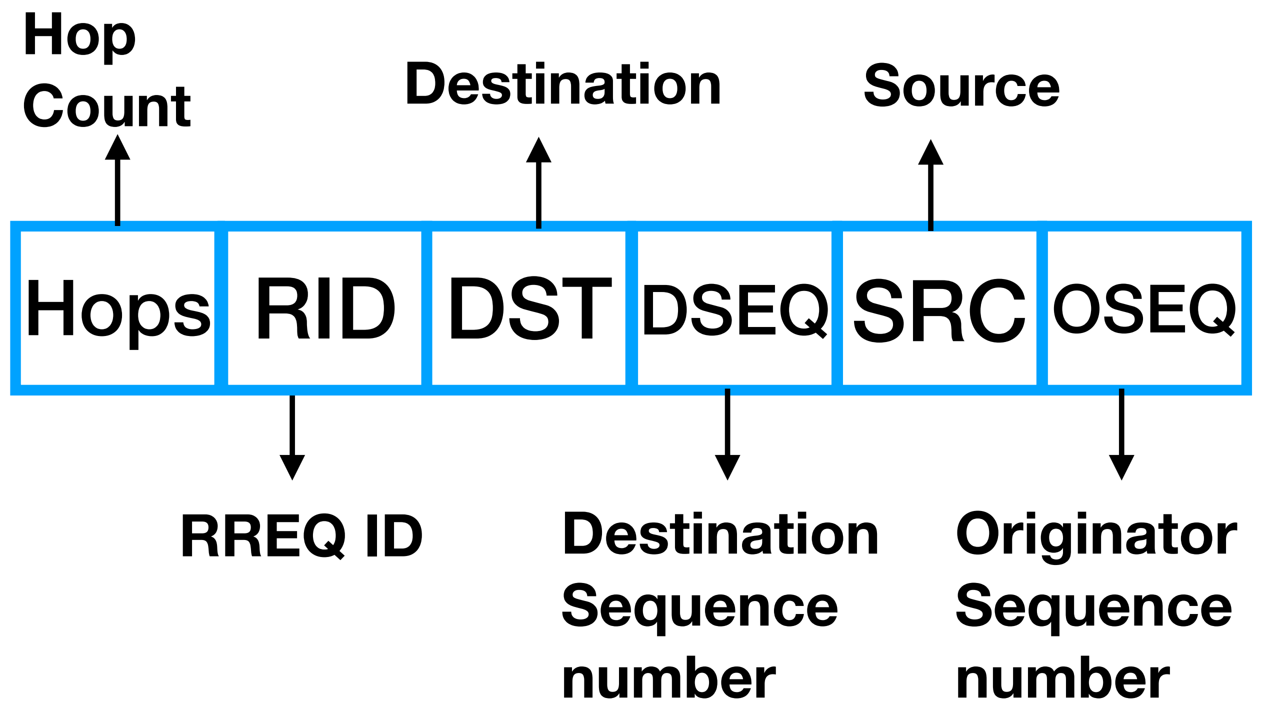
3 Ad Hoc On-demand Distance Vector - RREQ

Data packet arrives, destination is M, for which A does not have a route. Triggers RREQ

Assume A knows that M's sequence number is 3

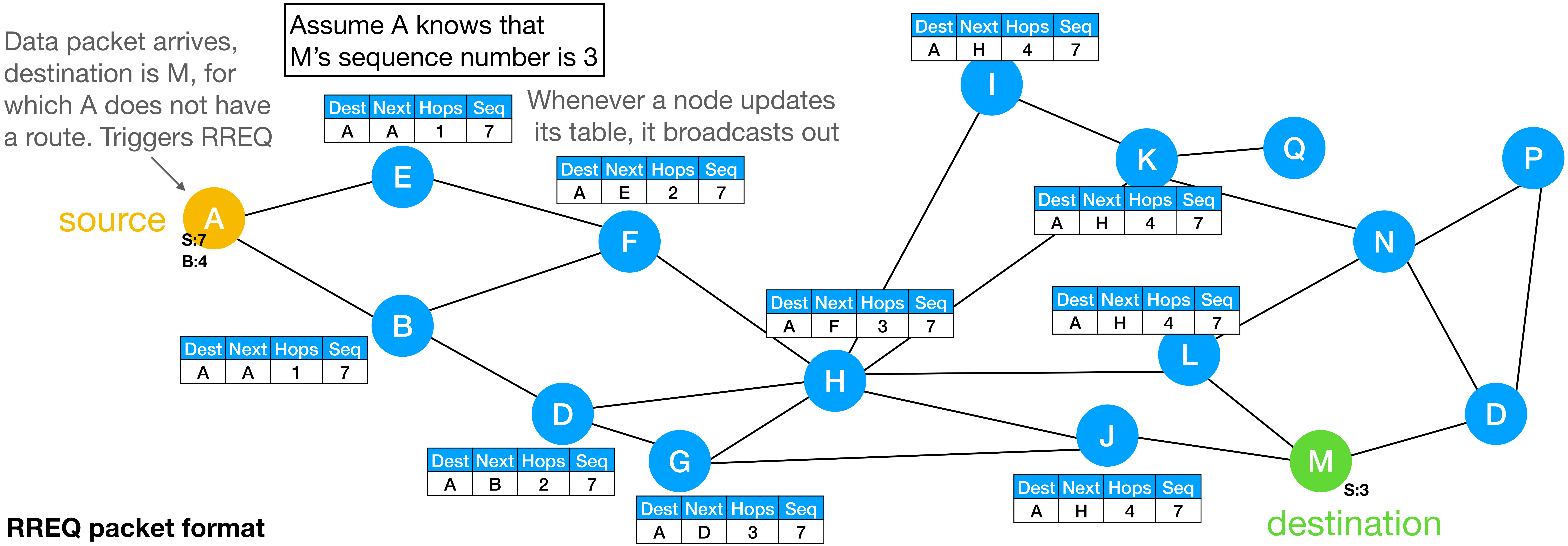


RREQ packet format

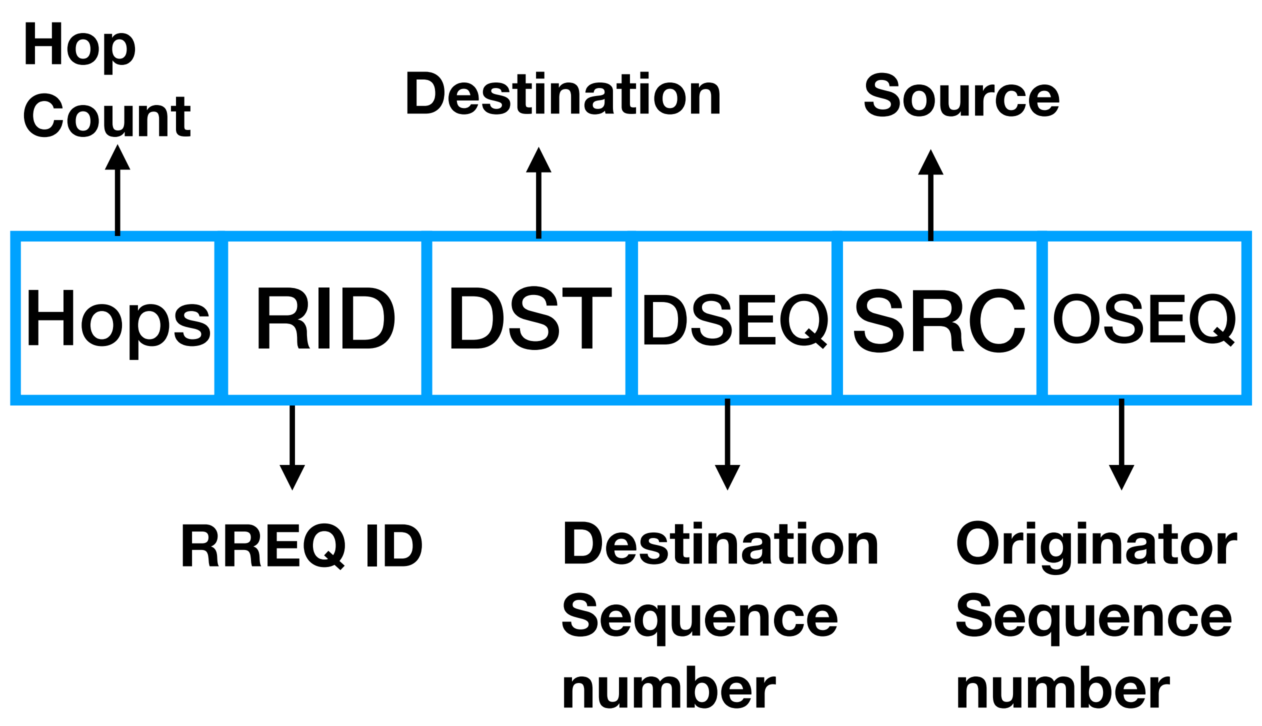


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

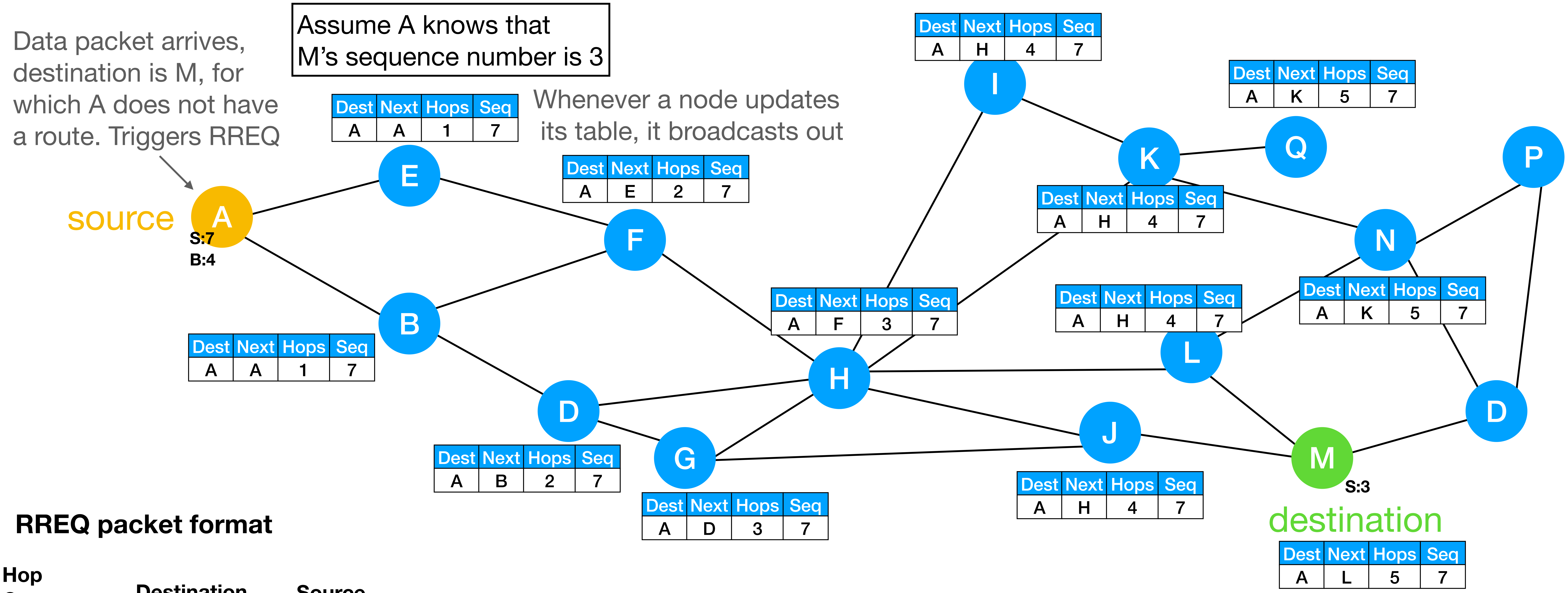


RREQ packet format



Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

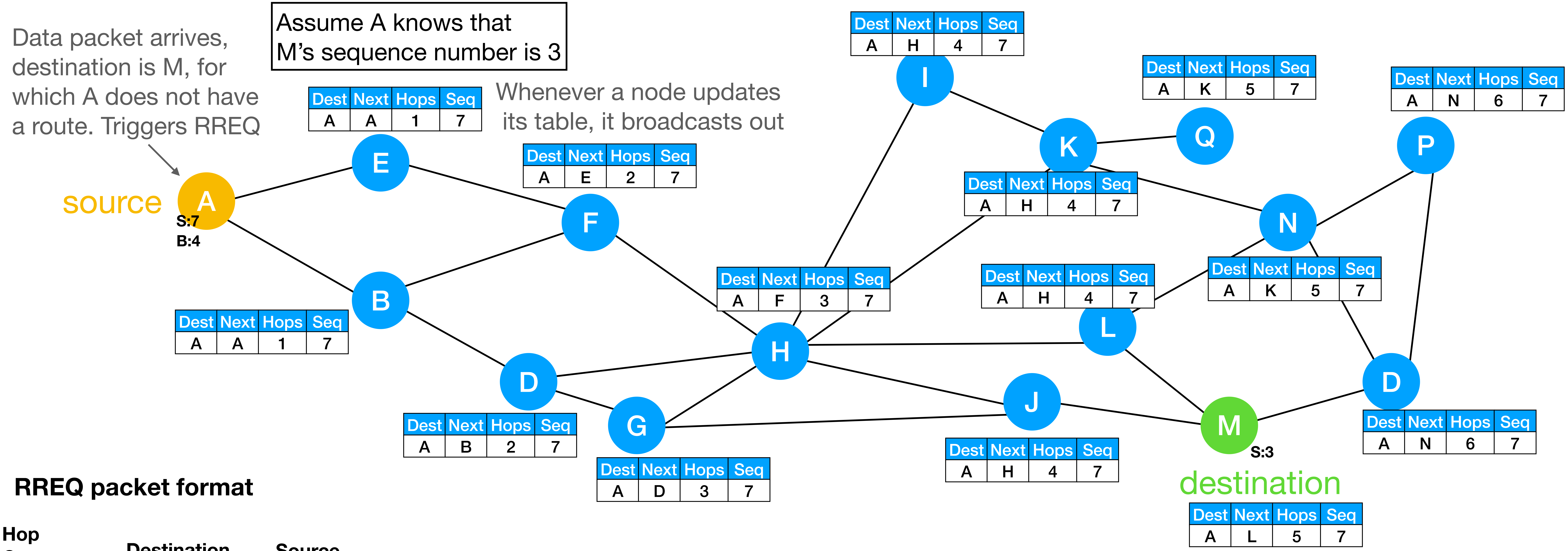


RREQ packet format

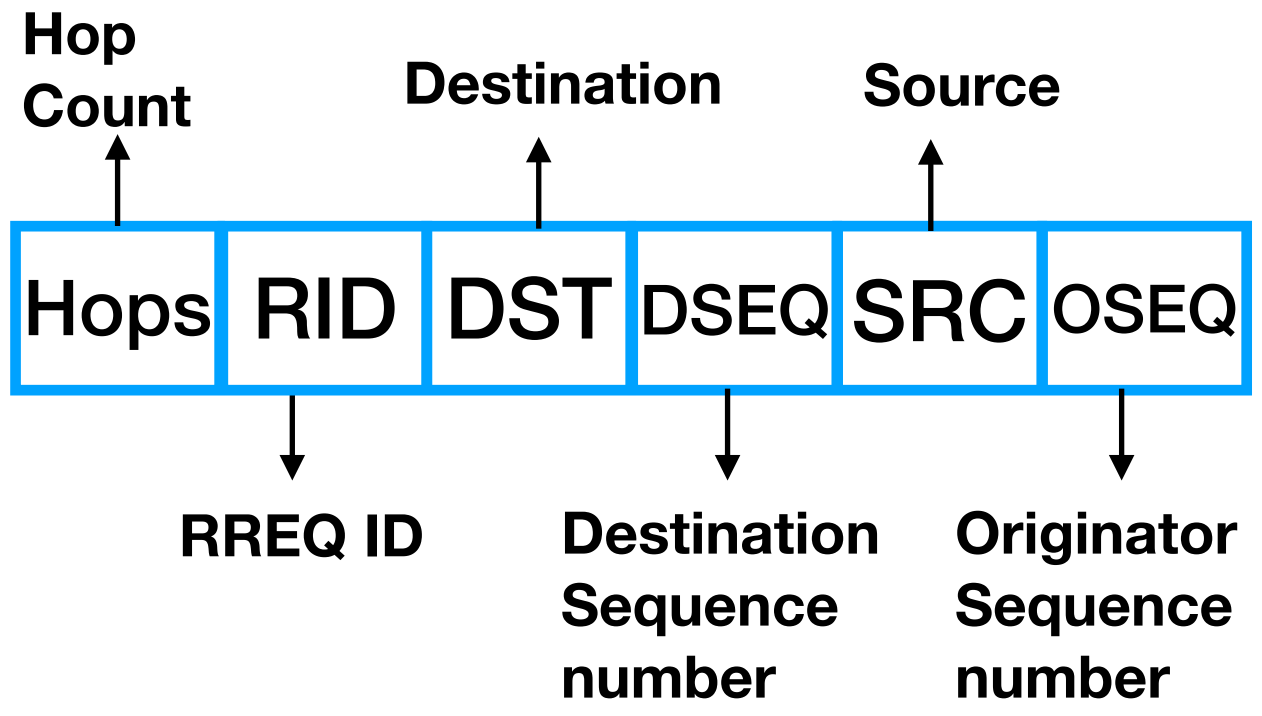
Hops	RID	DST	DSEQ	SRC	OSEQ
Hop Count	RREQ ID	Destination	Destination Sequence number	Source	Originator Sequence number

Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

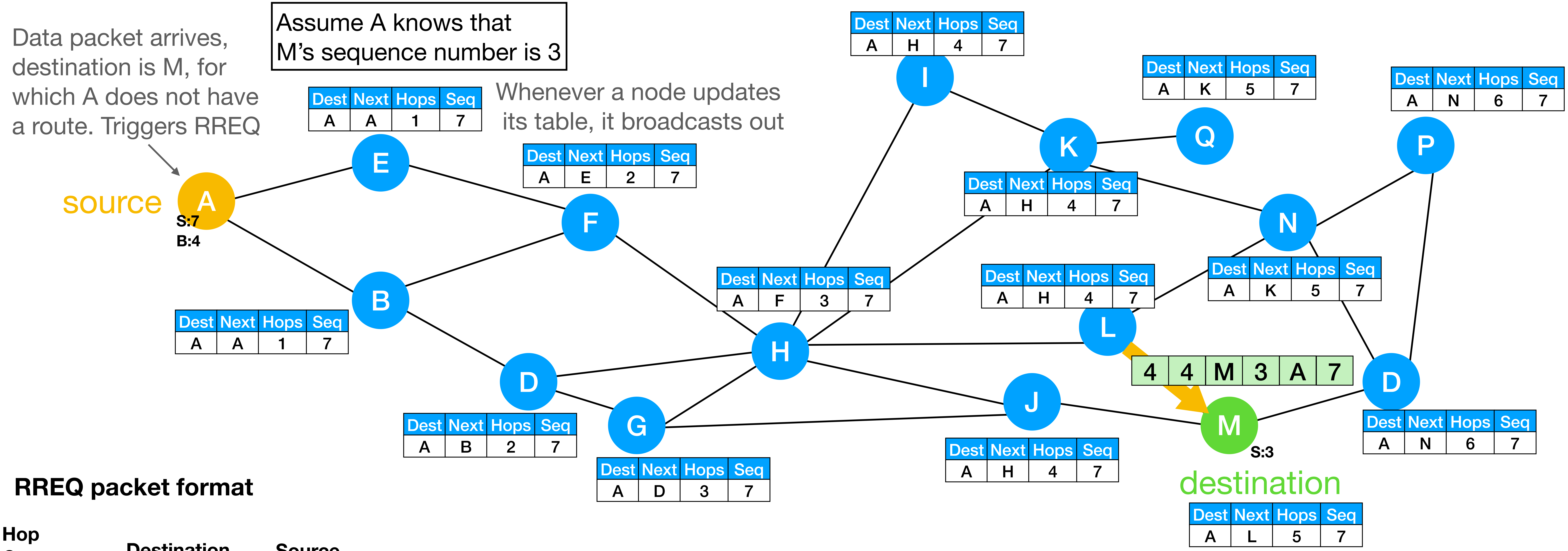


RREQ packet format

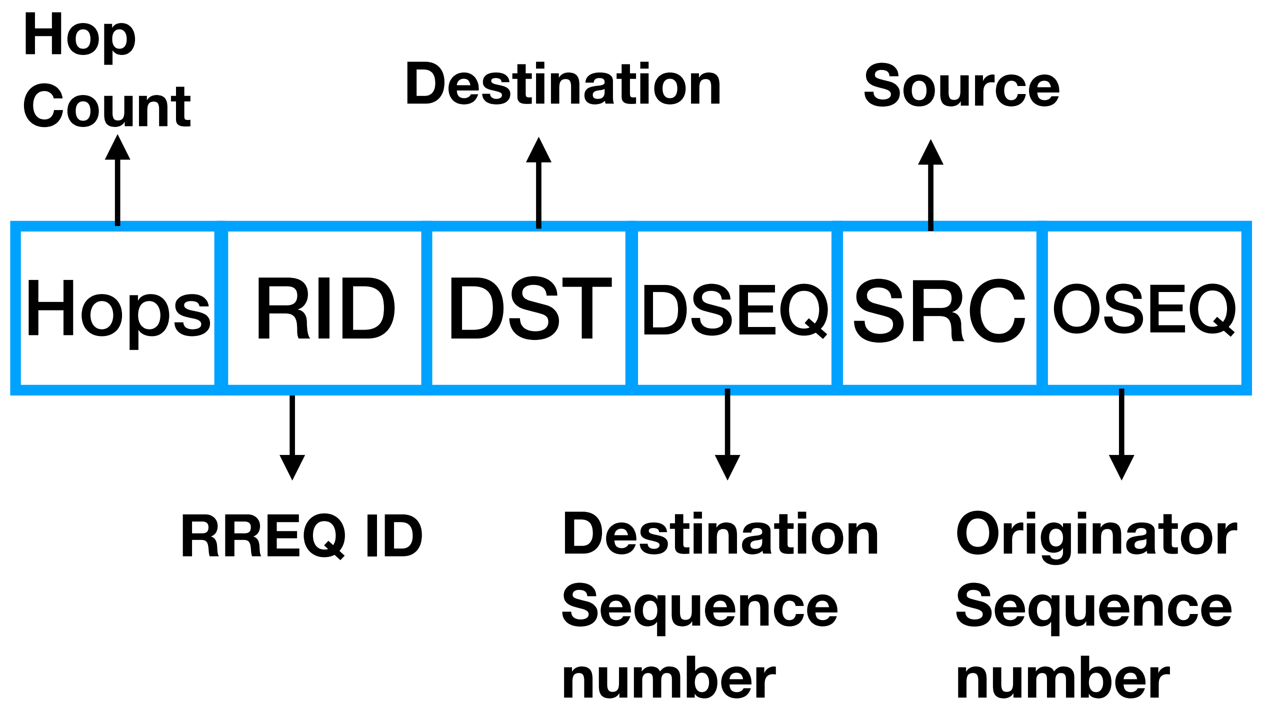


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREQ

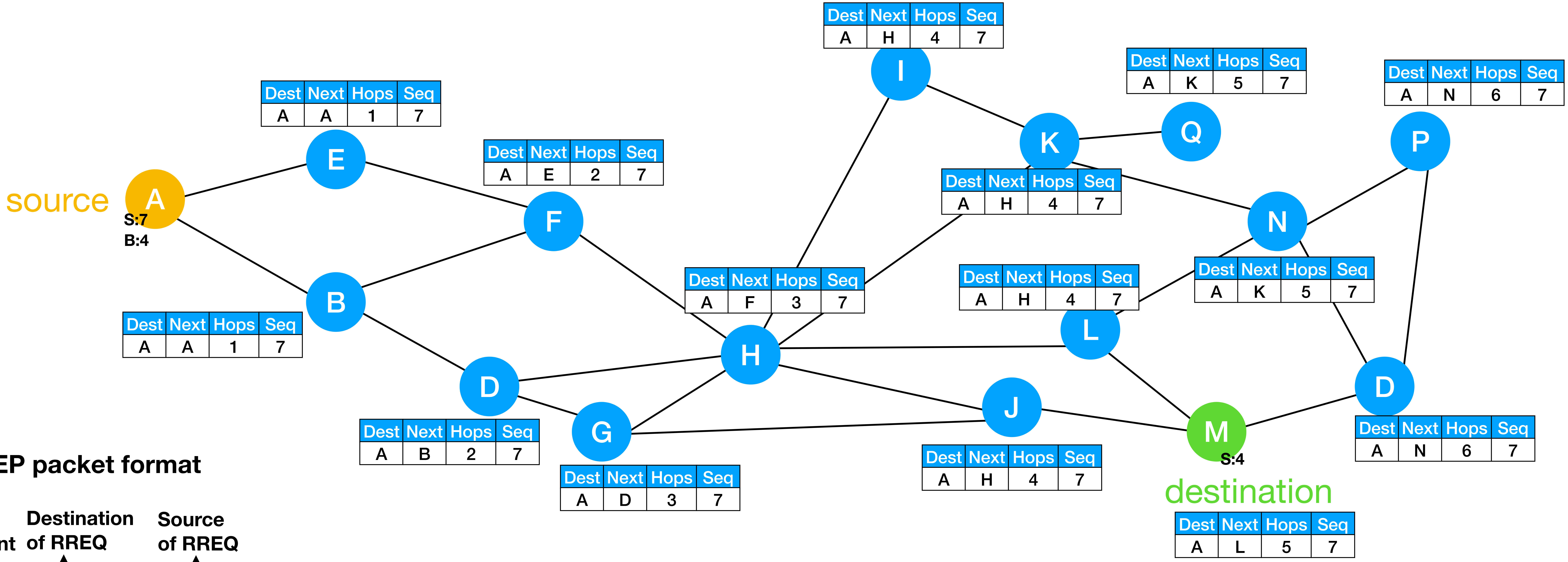


RREQ packet format

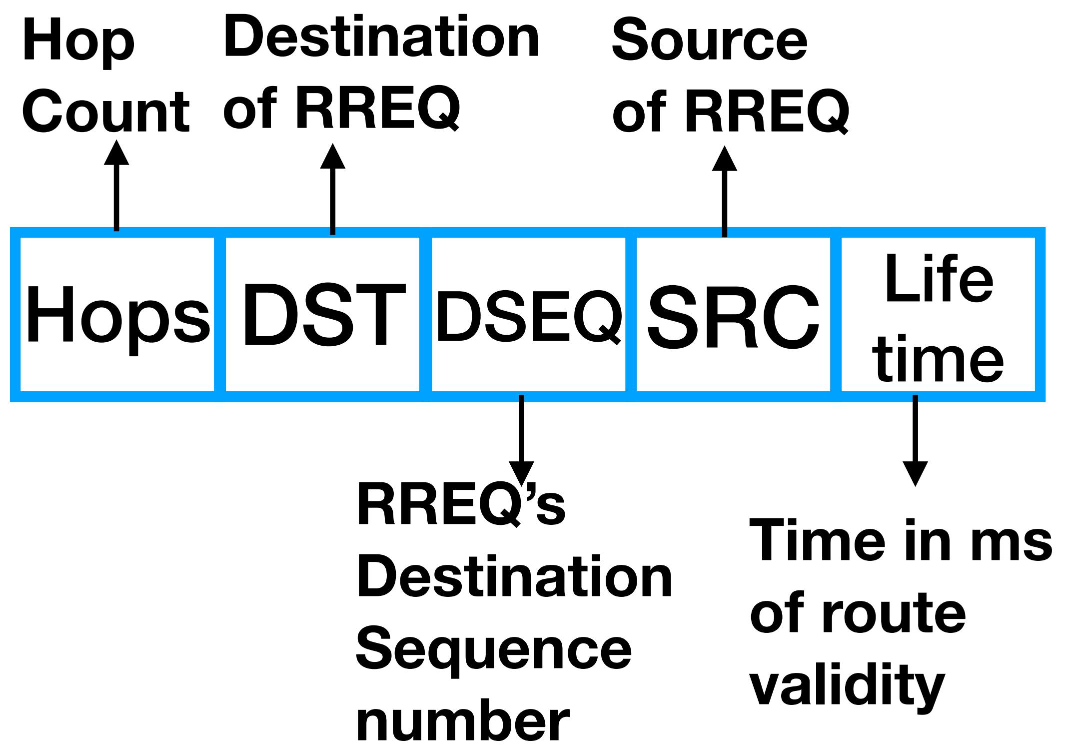


Step 1: source floods RREQ to find a path to destination.
Intermediate nodes maintain tables, created as in distance-vector, i.e., entry point back to source of the message

3 Ad Hoc On-demand Distance Vector - RREP

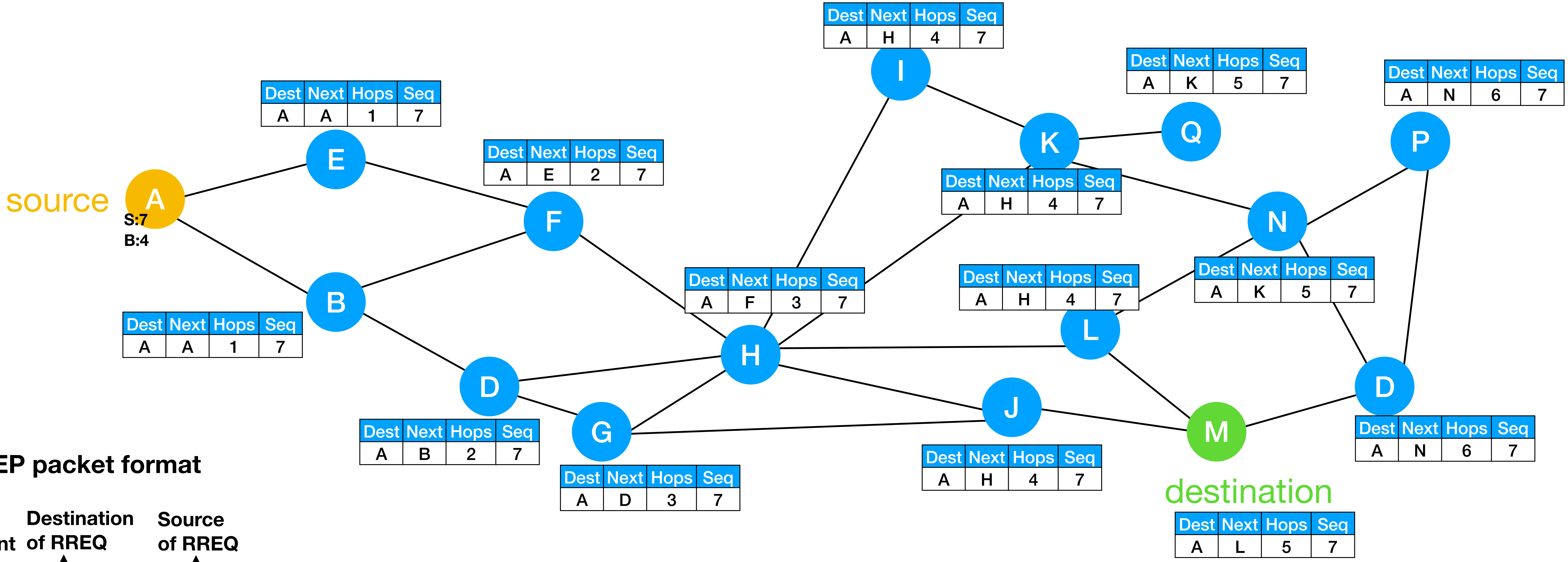


RREP packet format

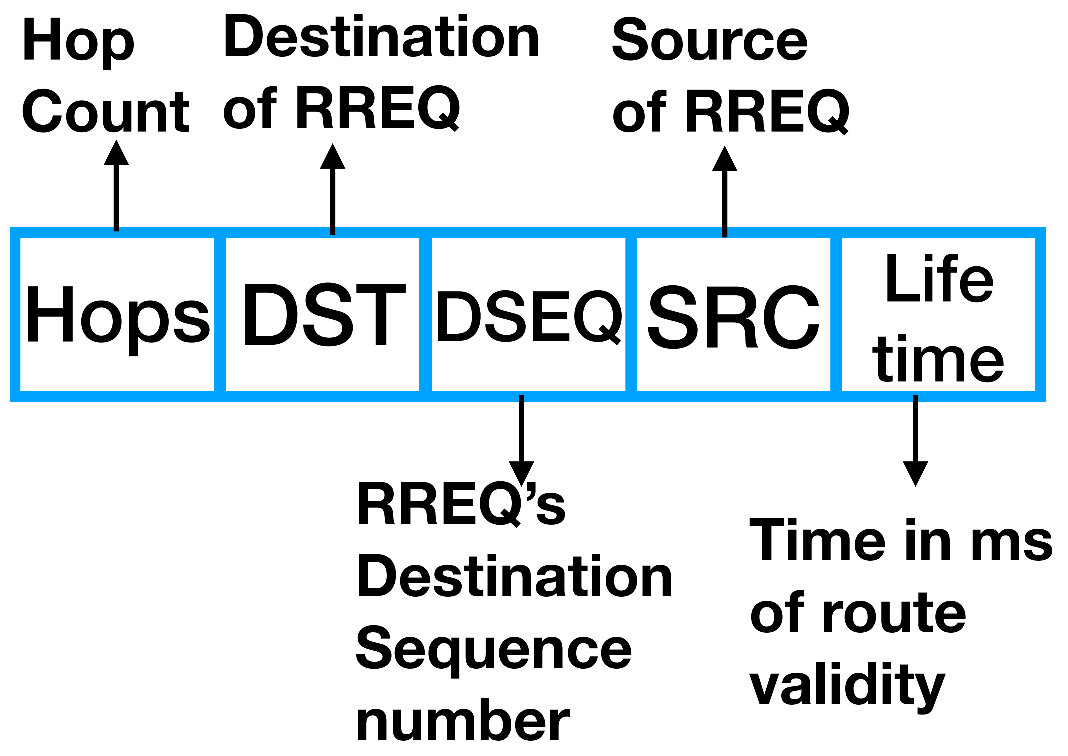


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP

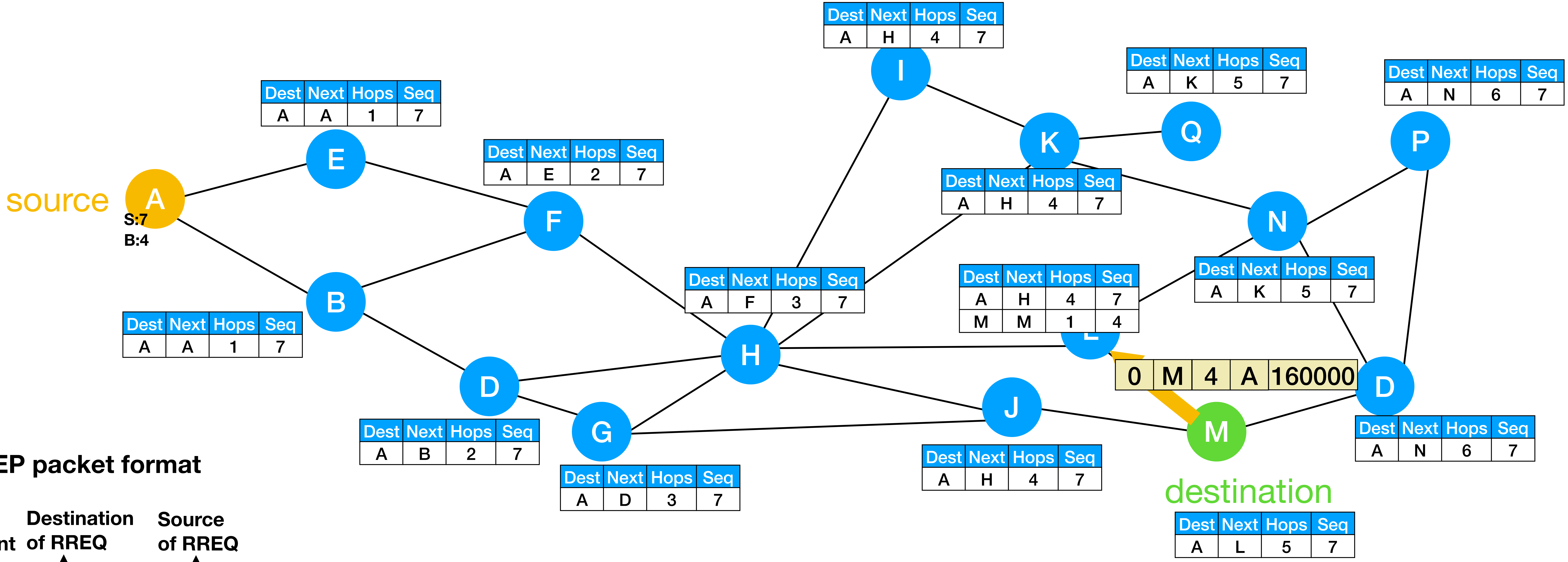


RREP packet format

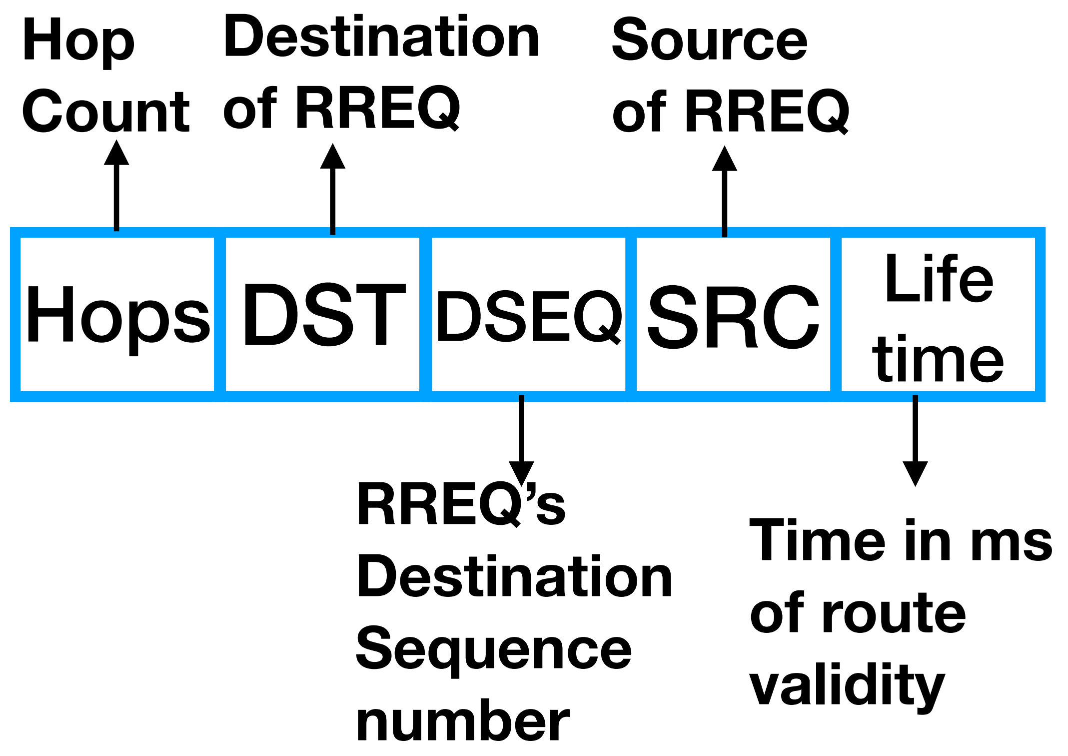


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP

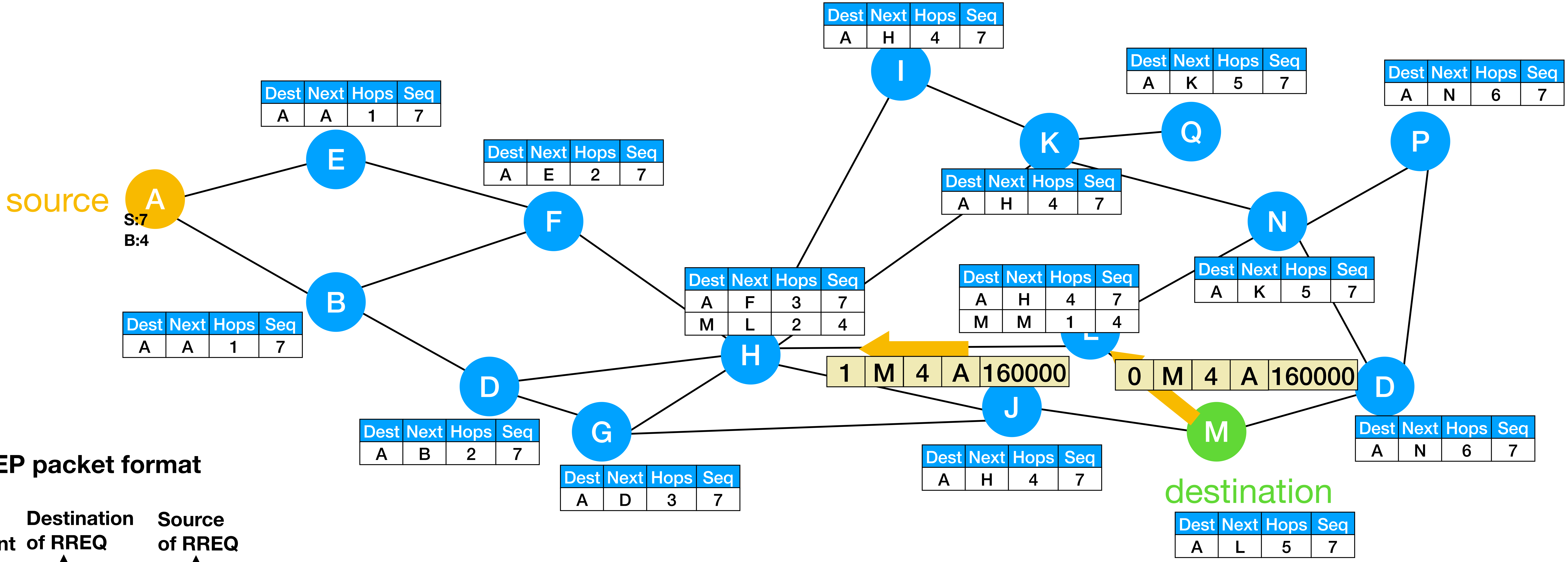


RREP packet format

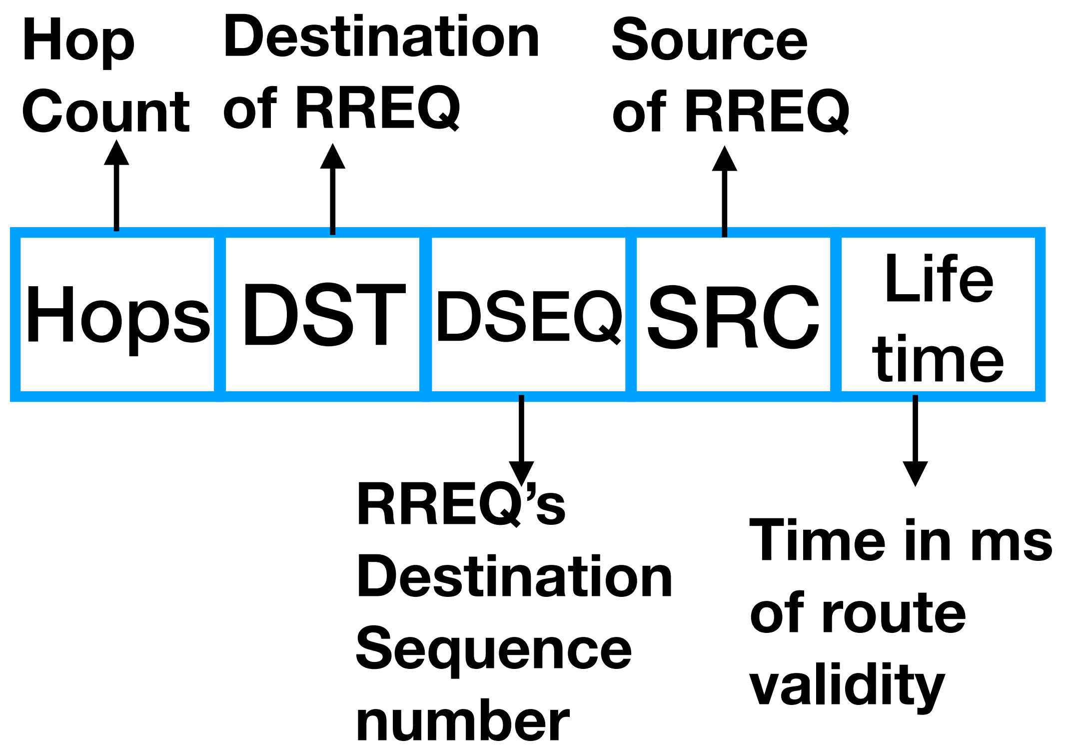


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP

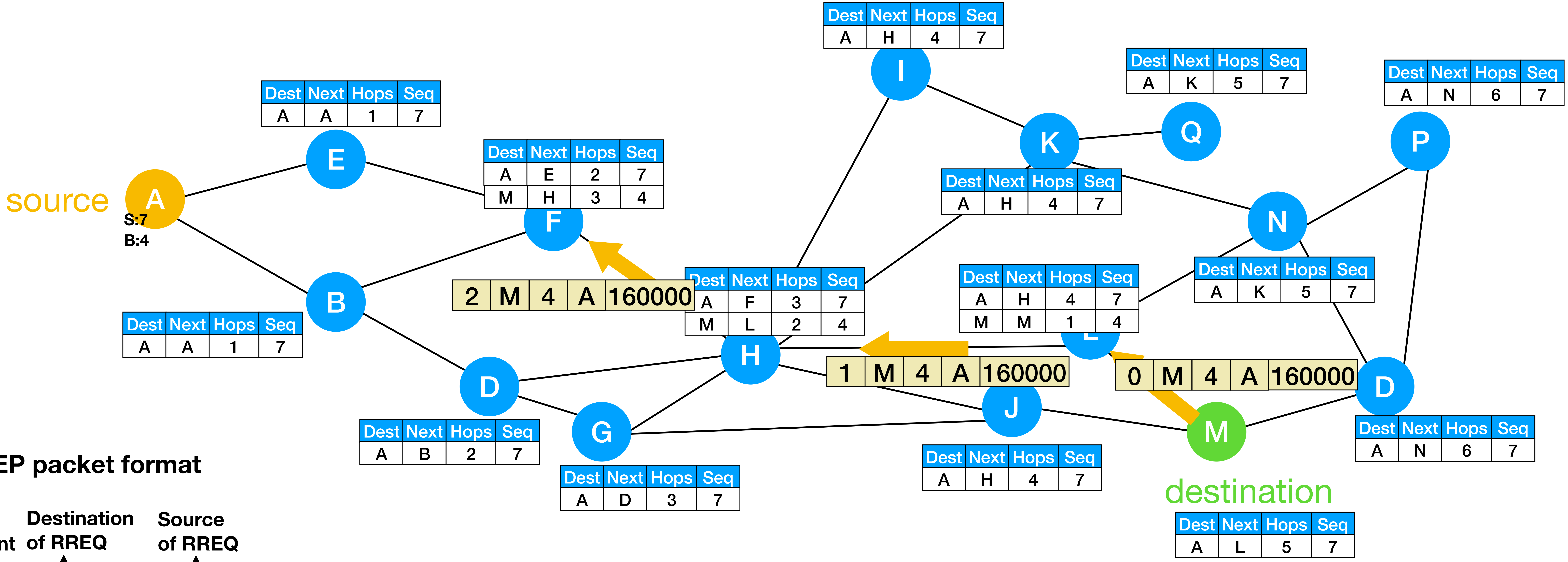


RREP packet format

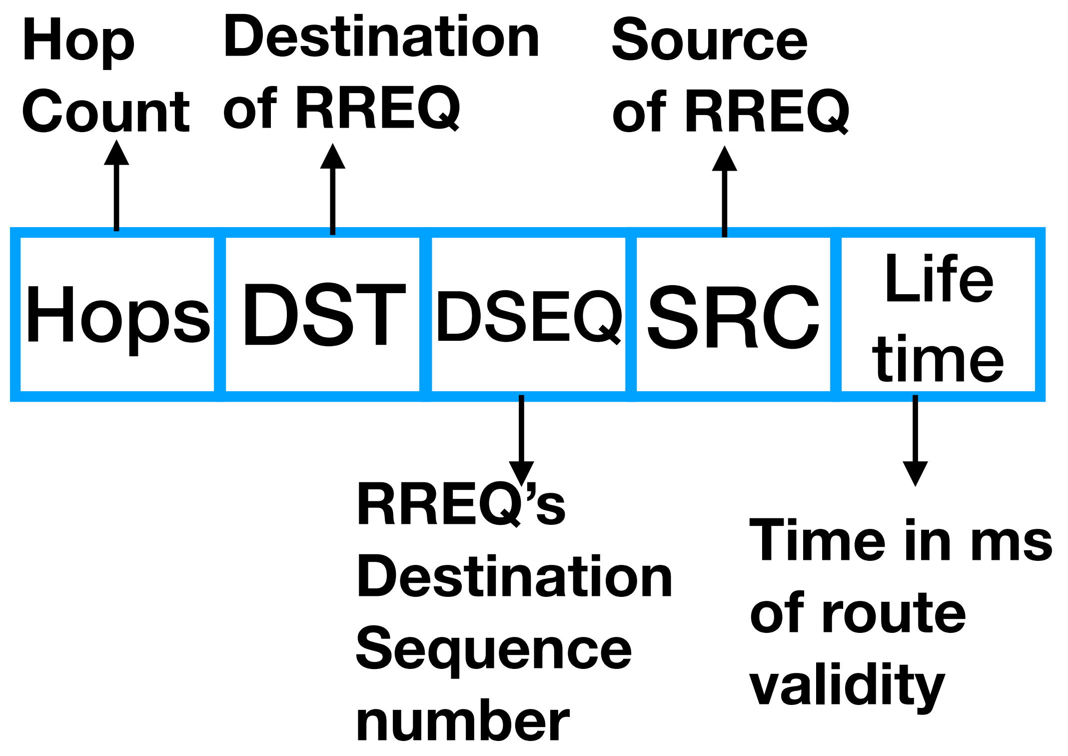


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP

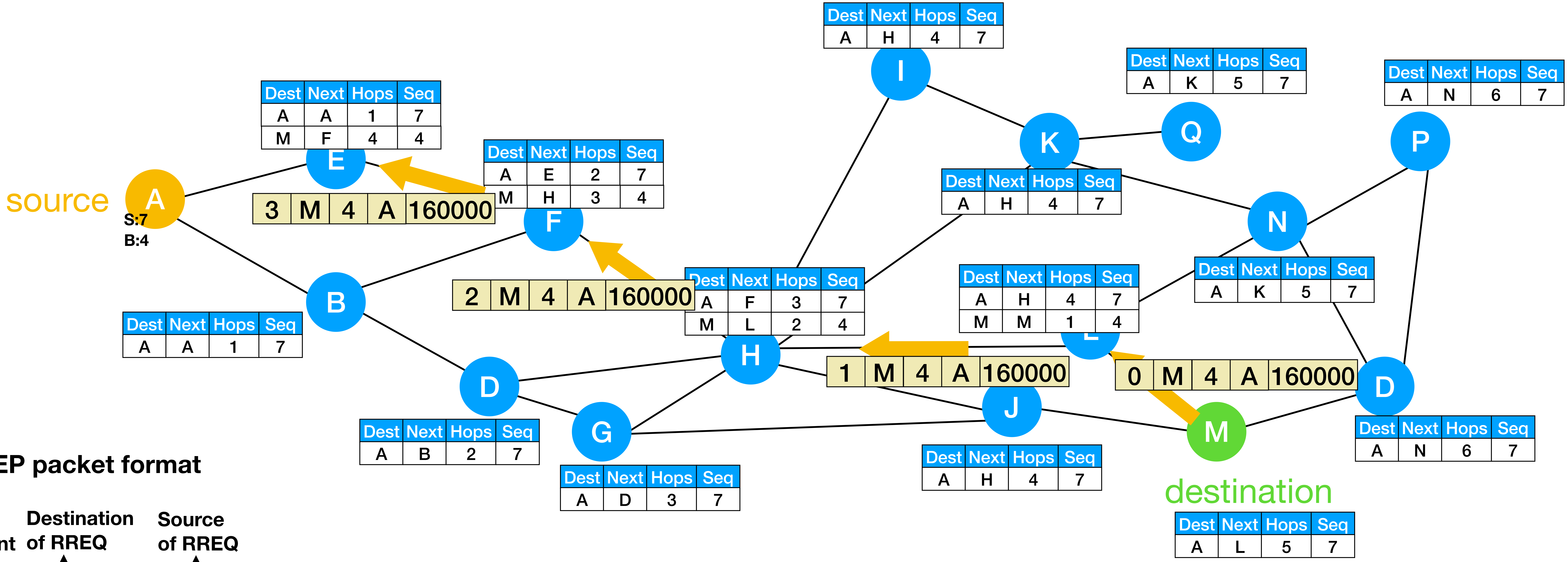


RREP packet format

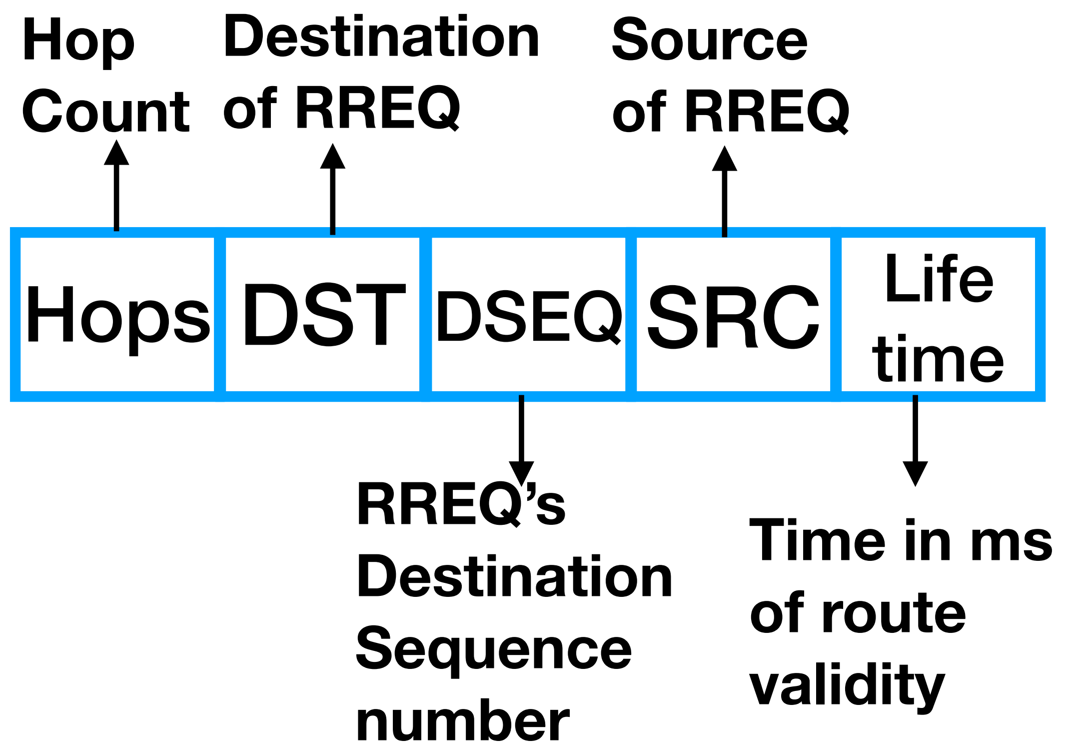


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP

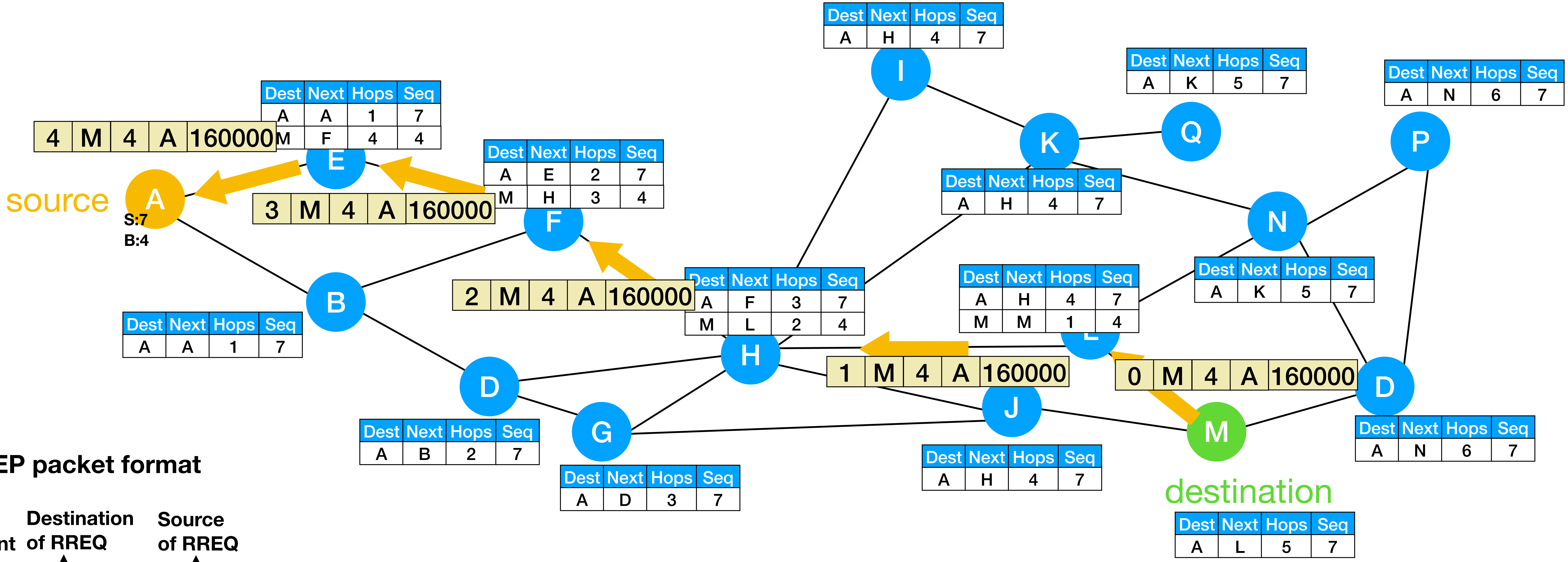


RREP packet format

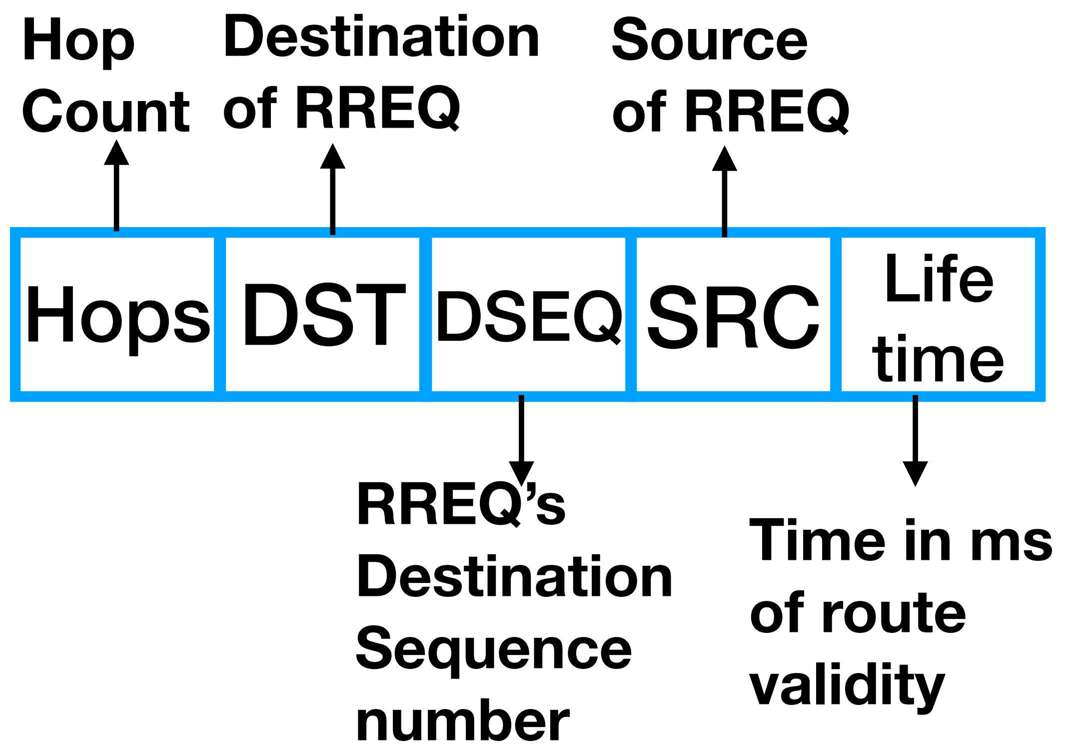


Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3 Ad Hoc On-demand Distance Vector - RREP



RREP packet format



Step 2: destination updates its sequence number and replies back through the trail of routing table entries created by the RREQ. Differently from classical distance-vector, where all nodes have all routes from everybody to everybody, the destination replies in unicast mode.

3

Ad Hoc On-demand Distance Vector - RERR

- Error messages are different from those of DSR. They are withdraw messages:
  - node detects a line break for next hop of a routing table entry
  - node receives data packet for which it has no route.
  - node receives RERR pertaining to one of node's active routes

- Nodes have the option to locally **repair** (initiate a broadcast and see if can route around failure)
  - instead of forwarding RERR, initiate RREQ for affected destination
  - can inflate path lengths over time

3

Ad Hoc On-demand Distance Vector (AODV) (2)

- Every route table entry at every node must include the latest information available about the destination sequence number.
  - it is updated every time a node receives new information about the sequence number from RREQ, RREP or RERR messages.

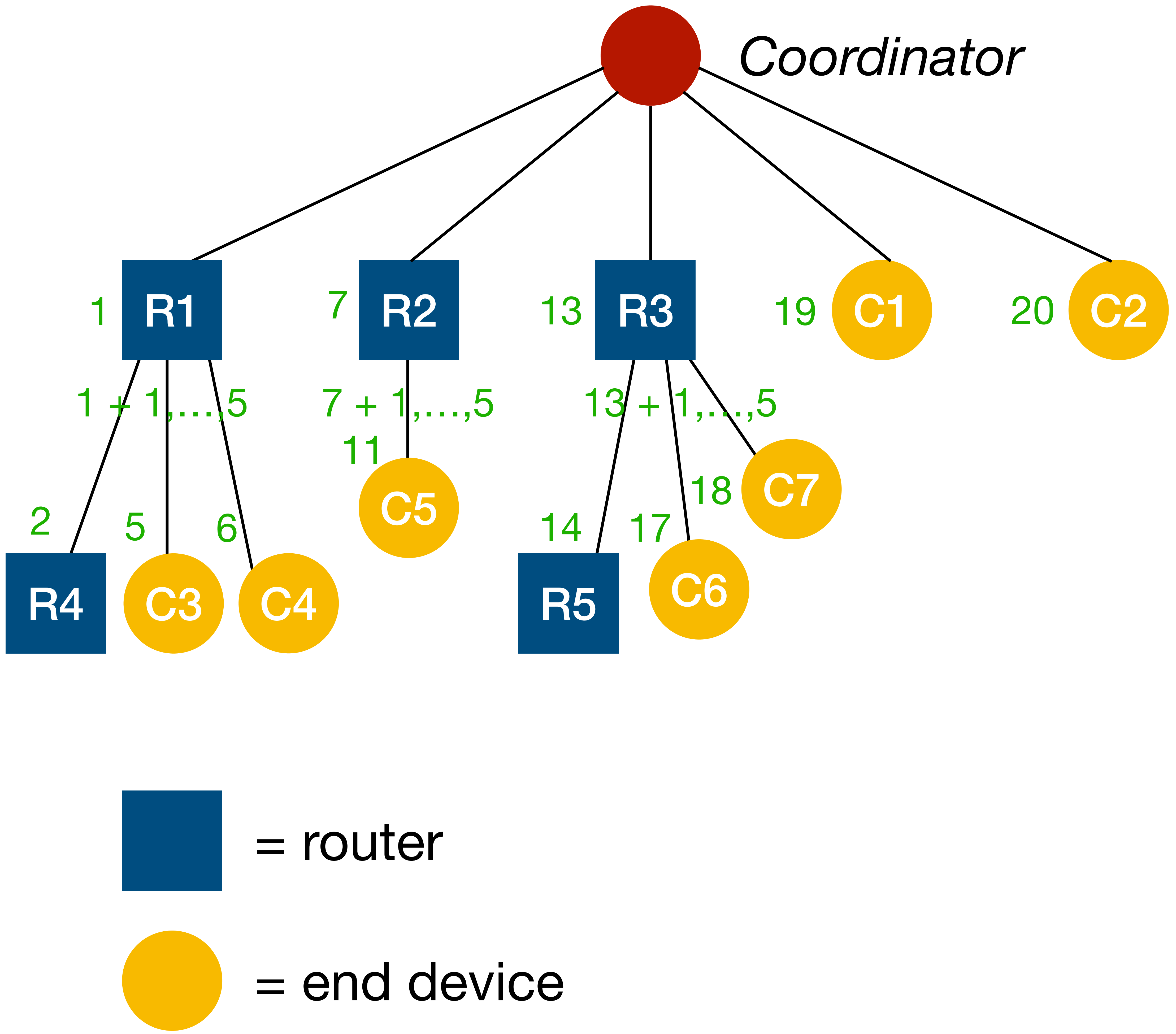
- Each node owns and maintains its sequence number to guarantee loop-freedom of all routes towards it. A node increases its sequence number:
  1. Before beginning a route discovery (i.e., before sending a RREQ message)
  2. Before originating a RREP.

Full documentation: https://www.rfc-editor.org/rfc/rfc3561

Reading

4

Hierarchical/Tree Routing



- Each node knows subrange of addresses for each children and is responsible for that block of addresses.
- A node getting a packet to send to some other nodes just needs to check if the address is in its children's subranges
  - if yes, sends to appropriate child
  - if no, sends to parent.
- Simple routing.

5

Geographic Routing (1)

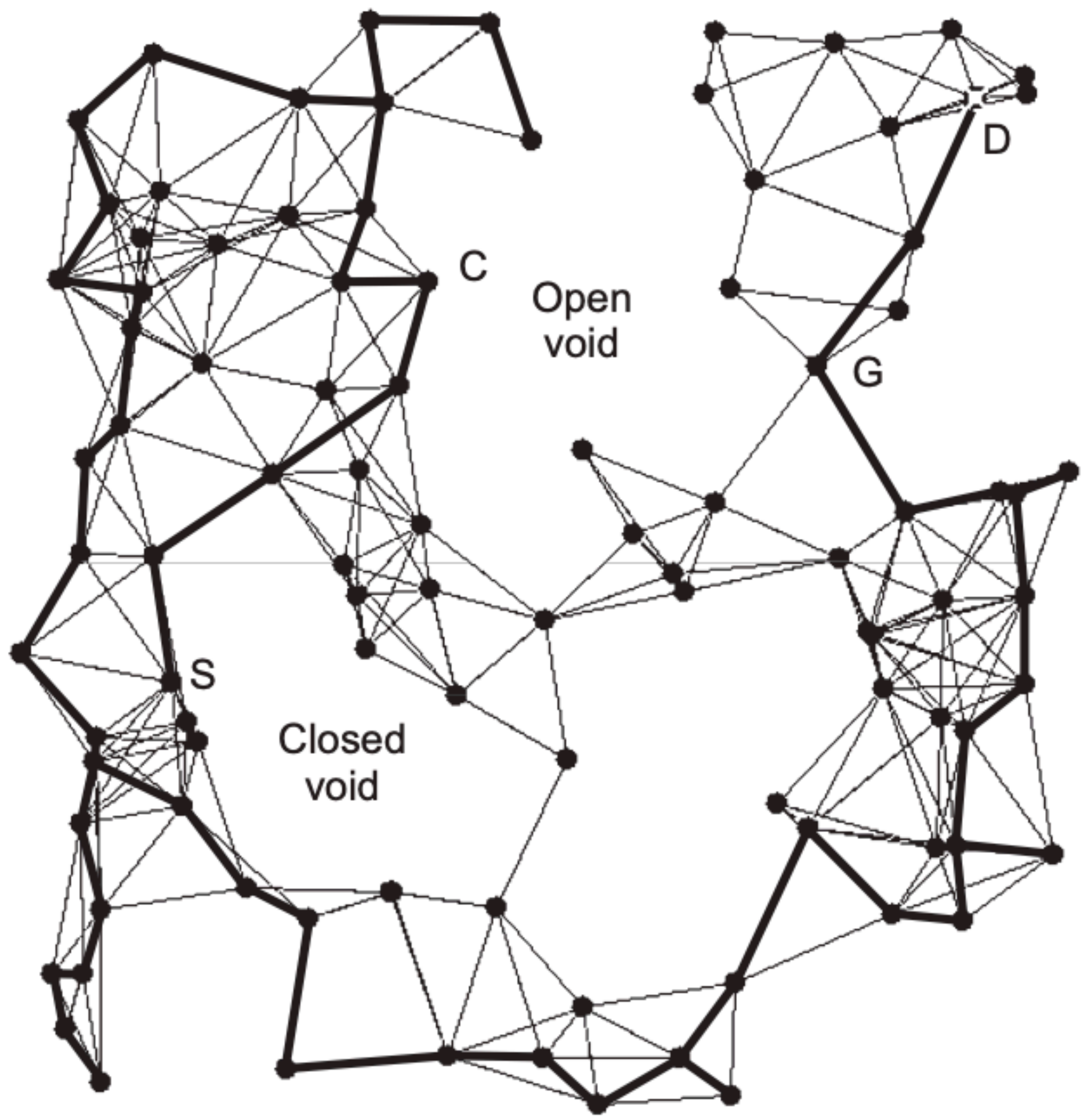
- Uses geographic address: uses geographic position information to make **progress** to destination.
- The source sends messages towards the geographic location of the destination.
- Each node keeps track of geographic location of neighbours, so it knows which neighbour makes most progress to destination.

5

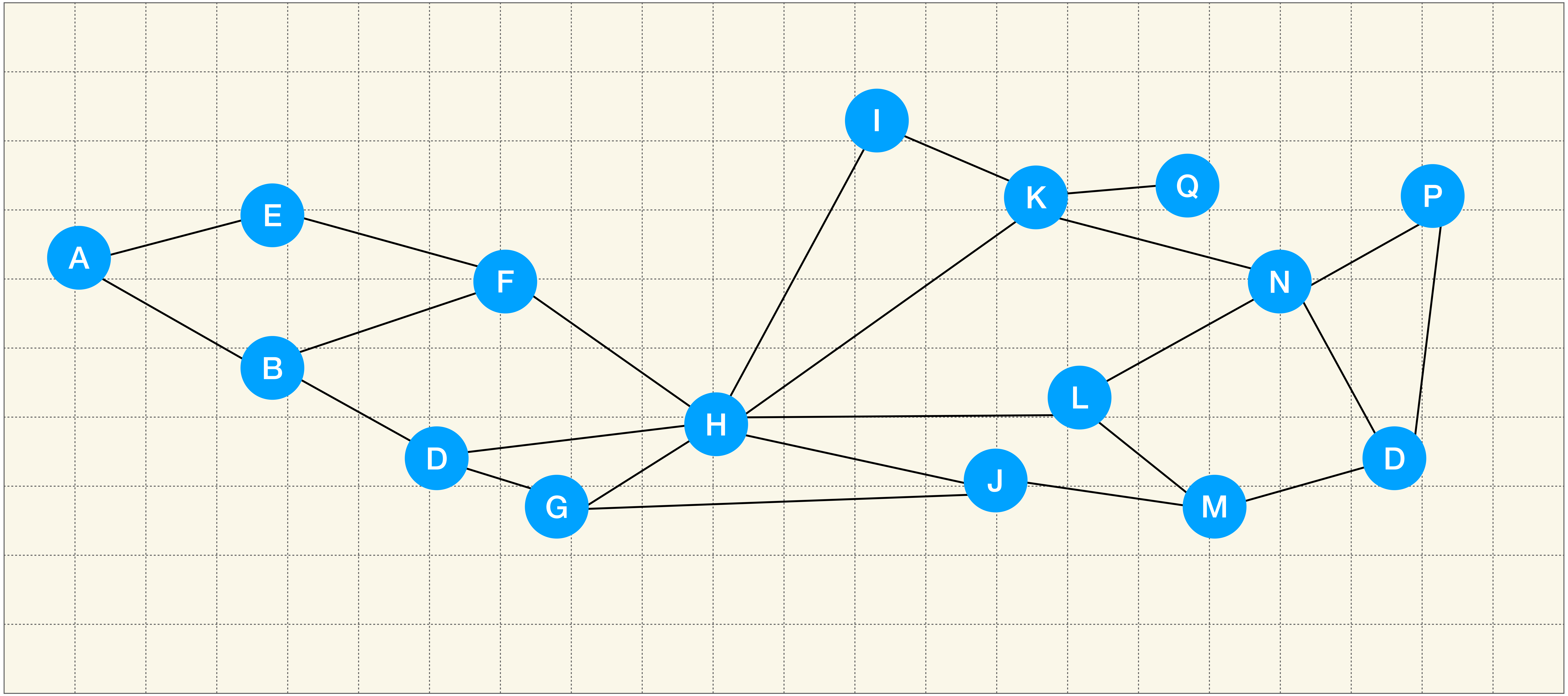
Geographic Routing (2)

- More complex than you might think
  - can get stuck in dead ends (“voids”, i.e., a node has no neighbours towards the destination)
  - can get stuck in loops (two neighbours think each other makes the most progress to the destination)

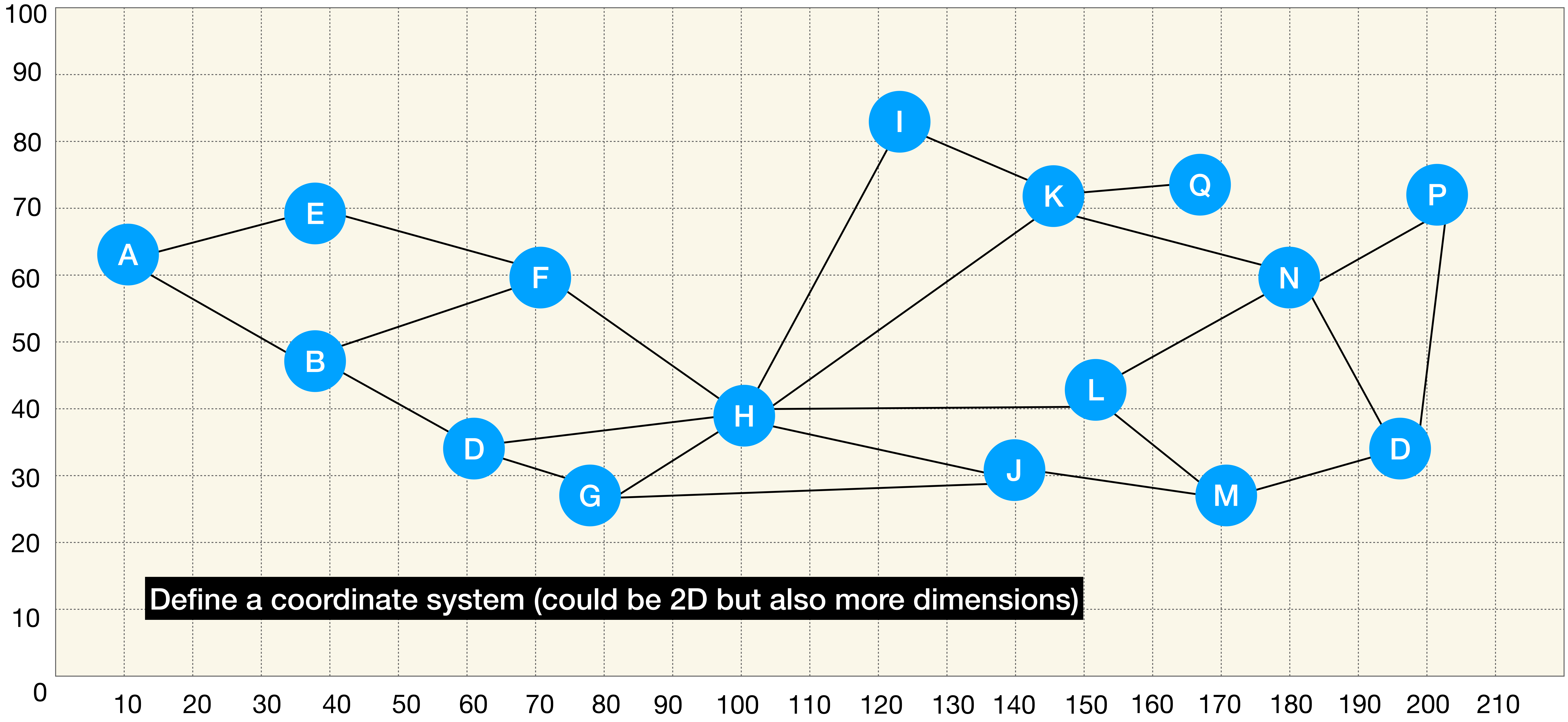
- Example of implementation of Geographic routing:
  - Greedy Forwarding



5 # Geographic Routing: Greedy Forwarding

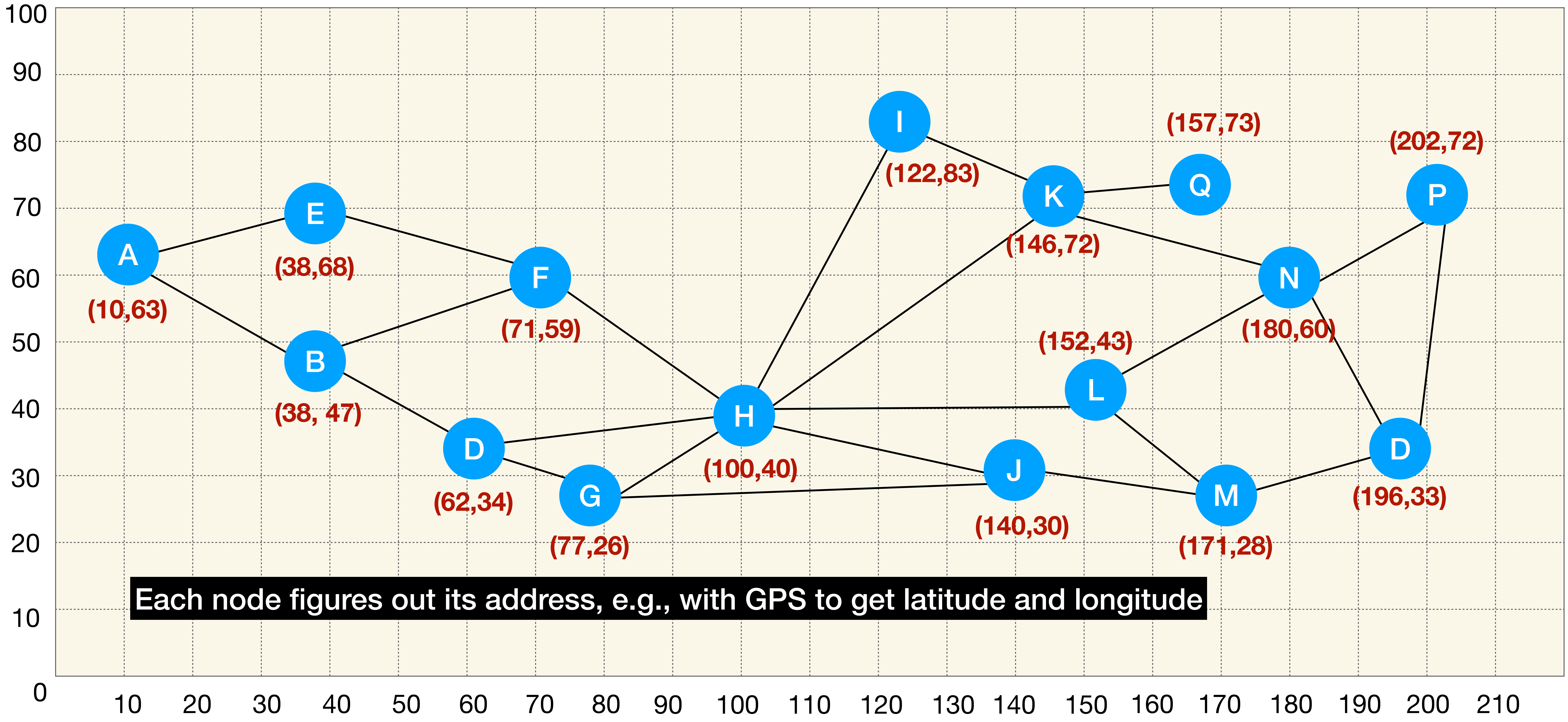


5 Geographic Routing: Greedy Forwarding

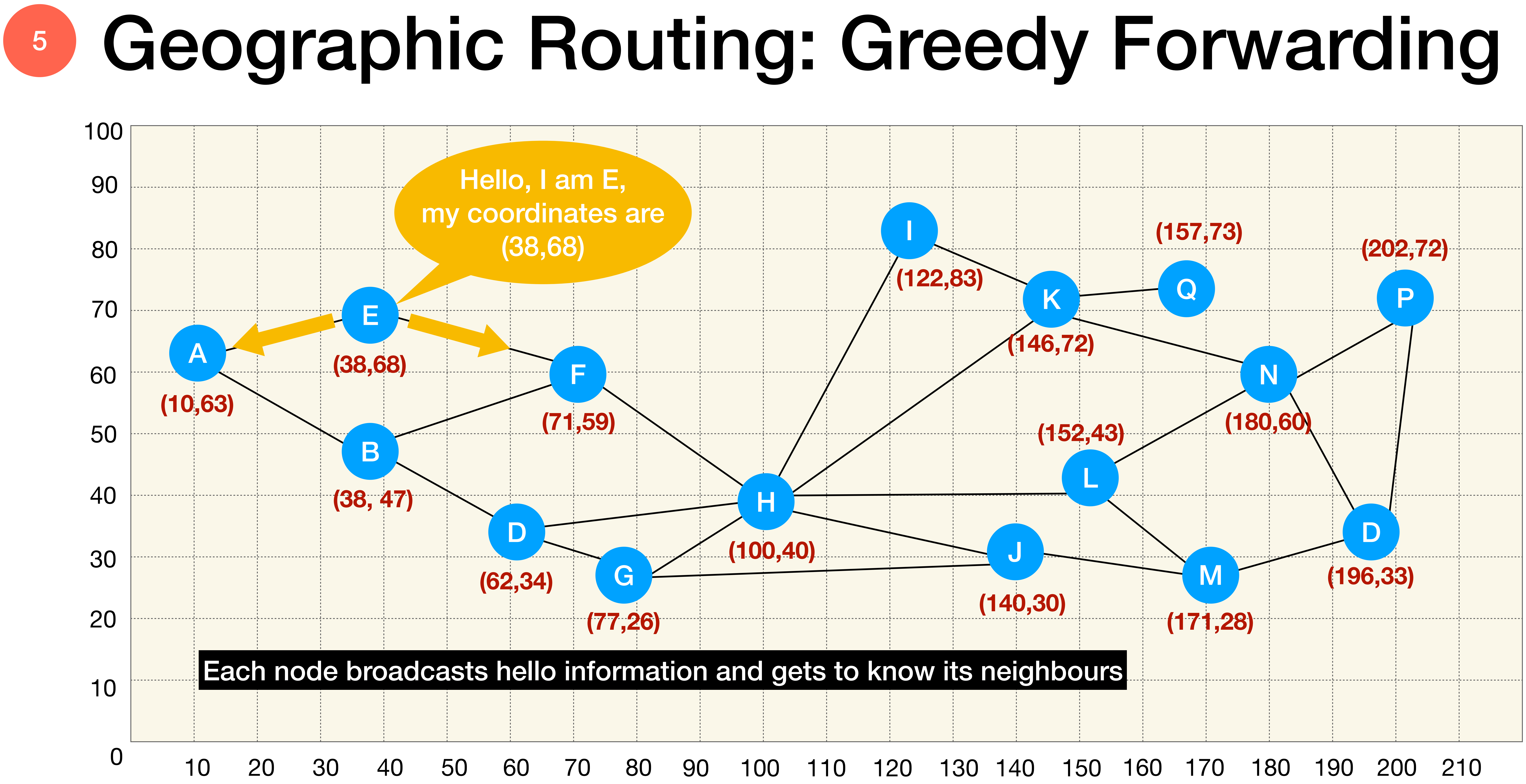


5

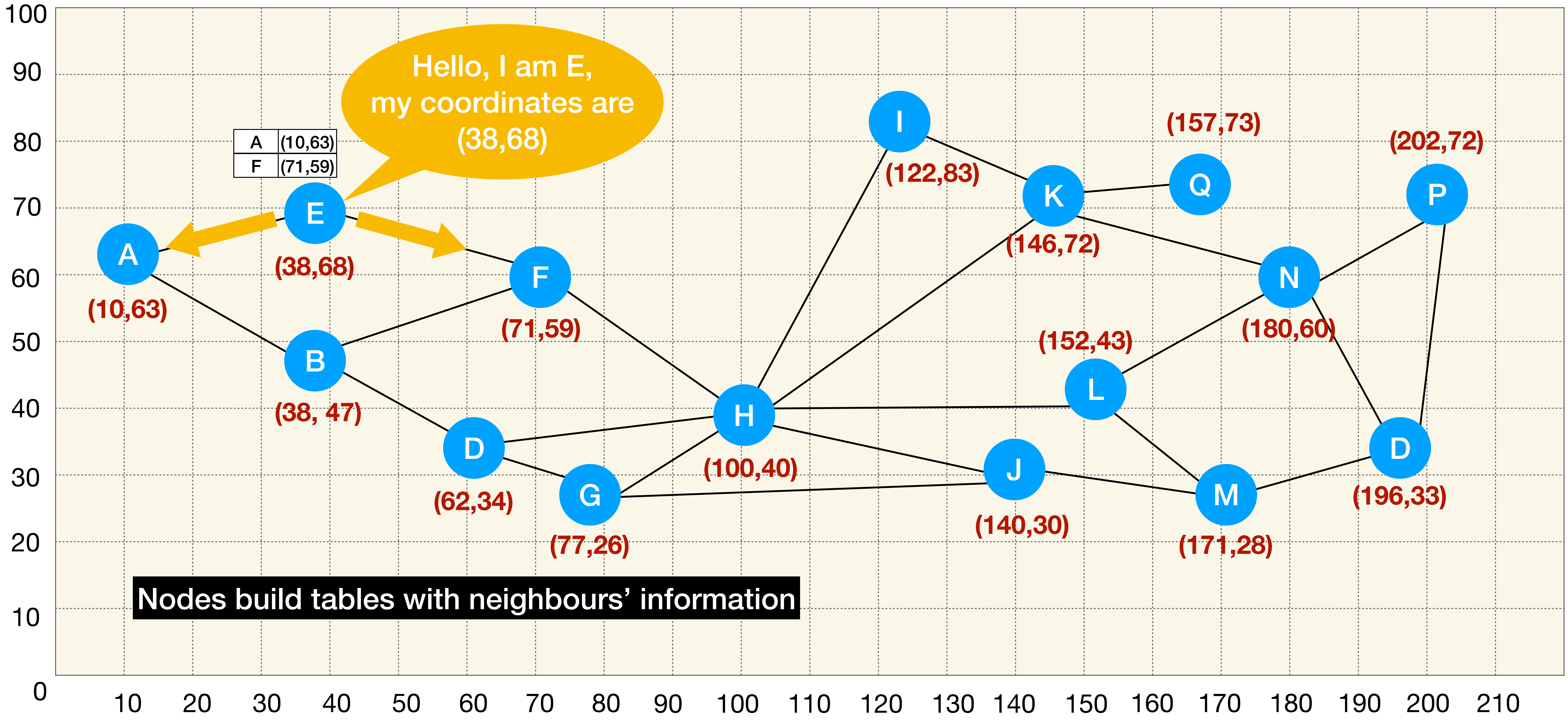
Geographic Routing: Greedy Forwarding



Geographic Routing: Greedy Forwarding

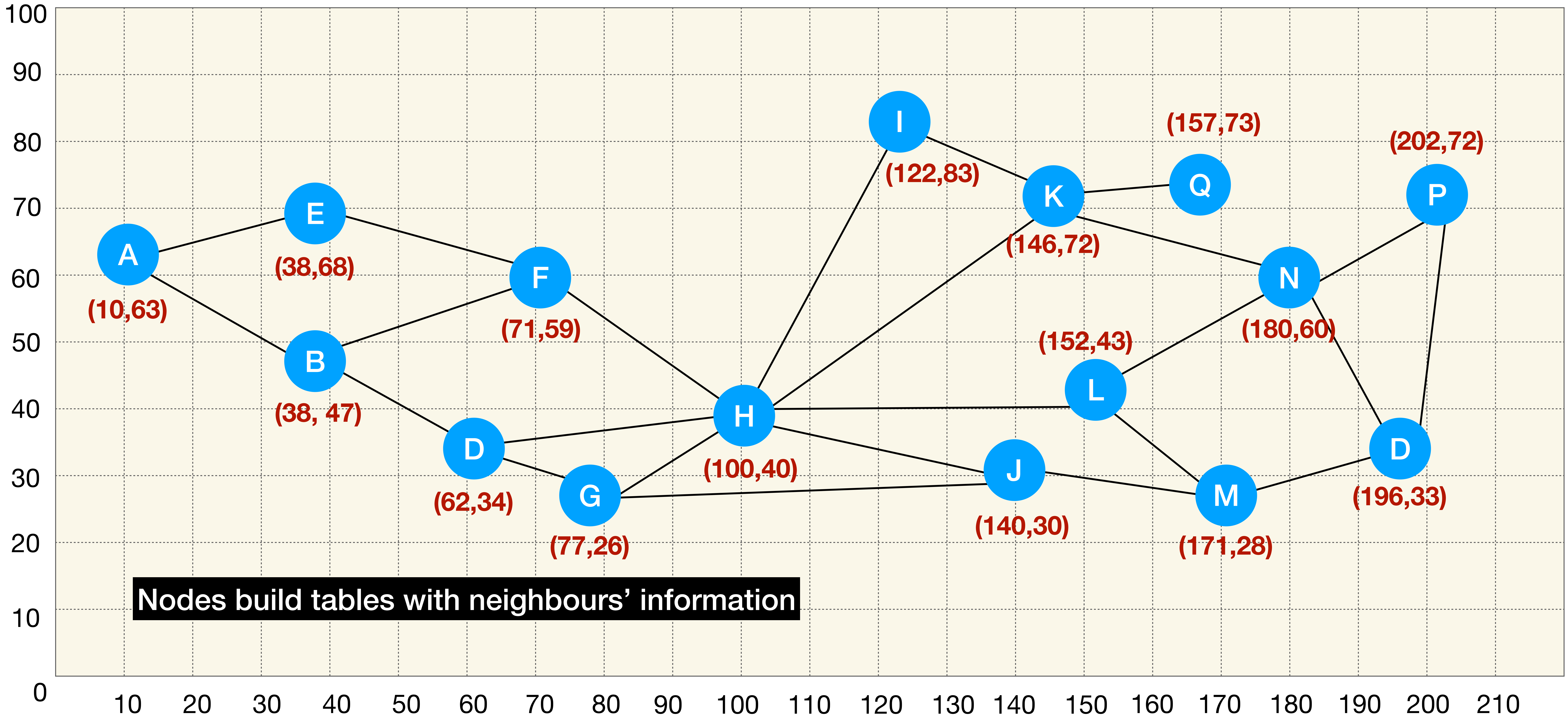


5 Geographic Routing: Greedy Forwarding



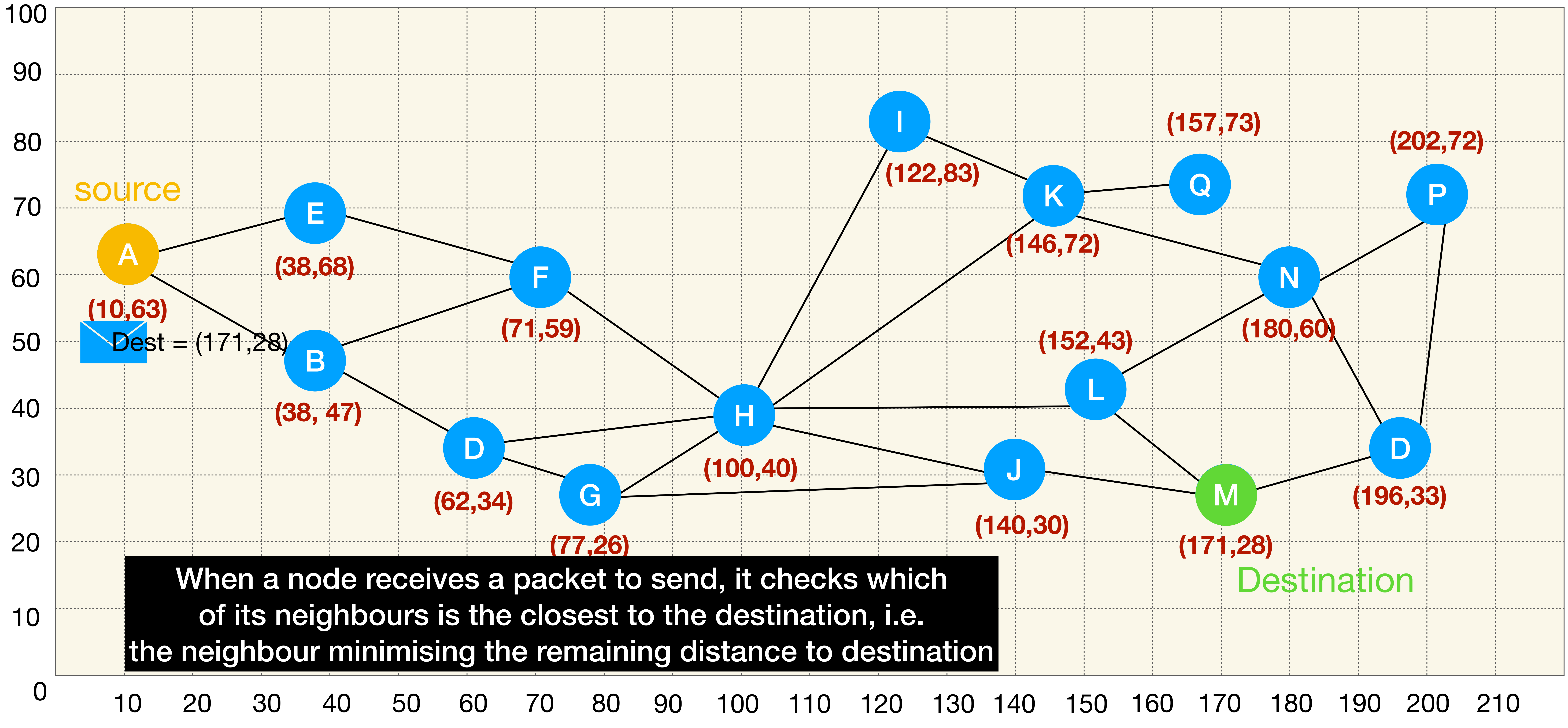
5

Geographic Routing: Greedy Forwarding



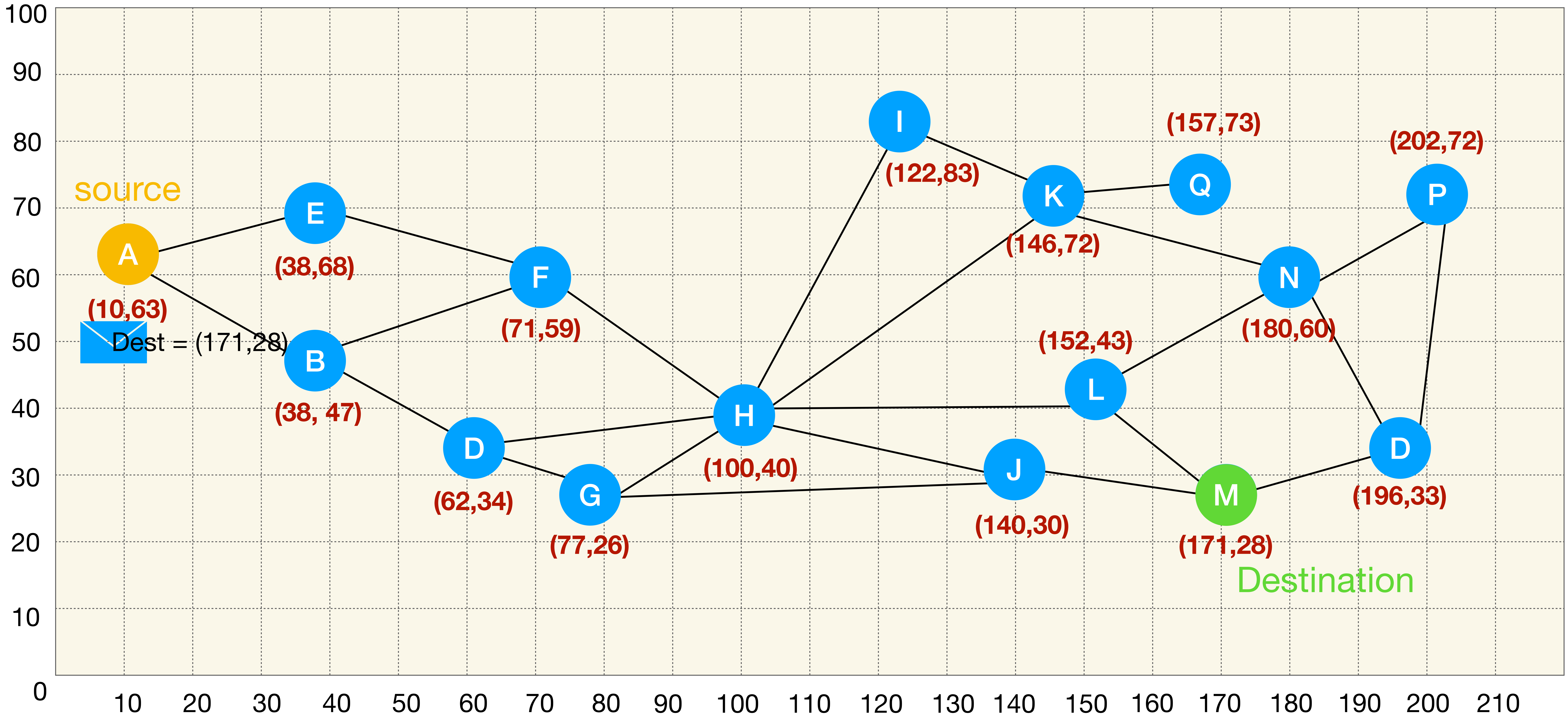
5

Geographic Routing: Greedy Forwarding



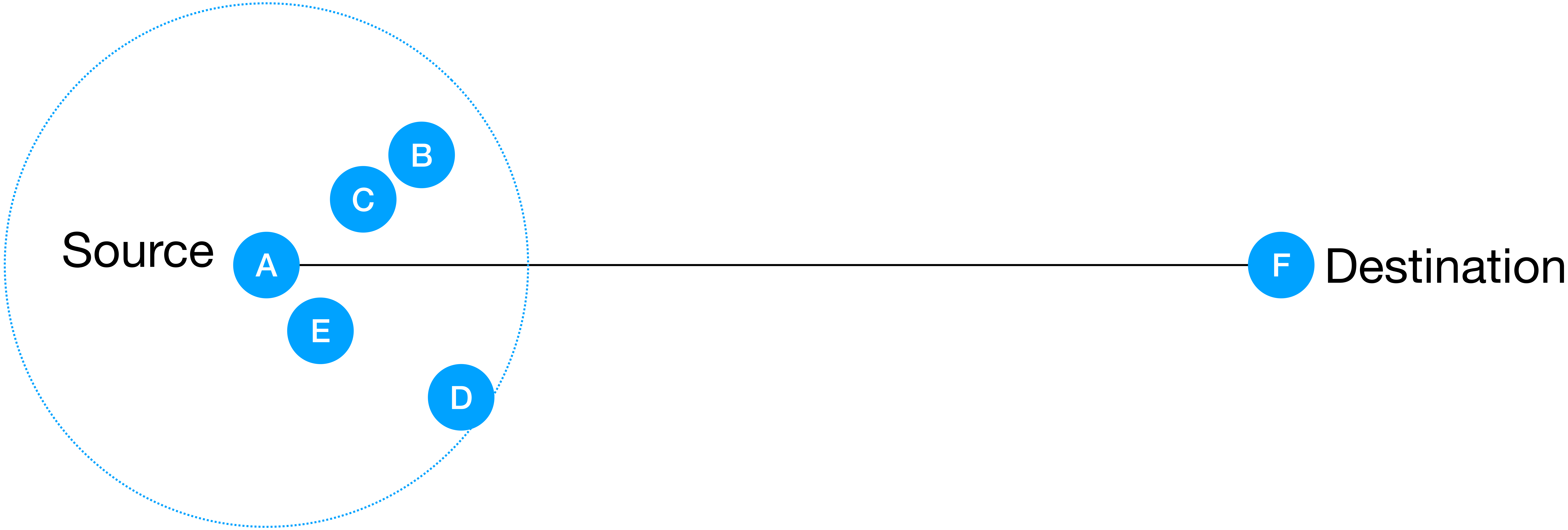
When a node receives a packet to send, it checks which of its neighbours is the closest to the destination, i.e. the neighbour minimising the remaining distance to destination

Geographic Routing: Greedy Forwarding



5

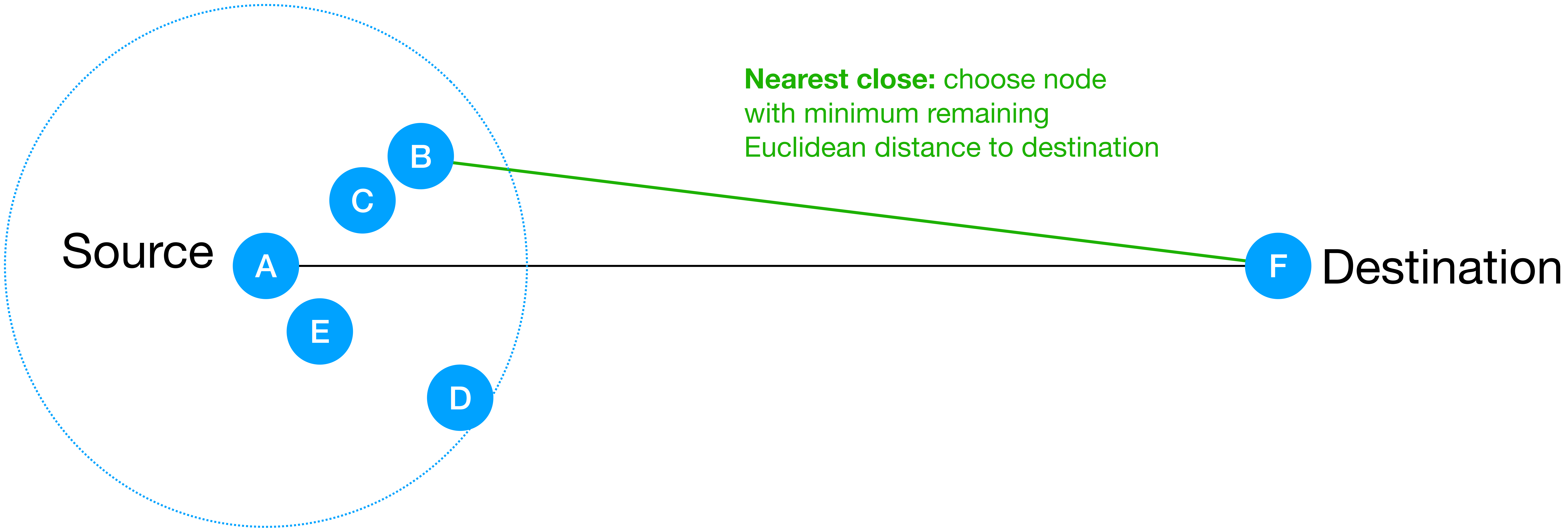
Geographic Routing: different distance metrics for Greedy Forwarding



- Different ways for measuring distances, and therefore, next hop to forward data to.

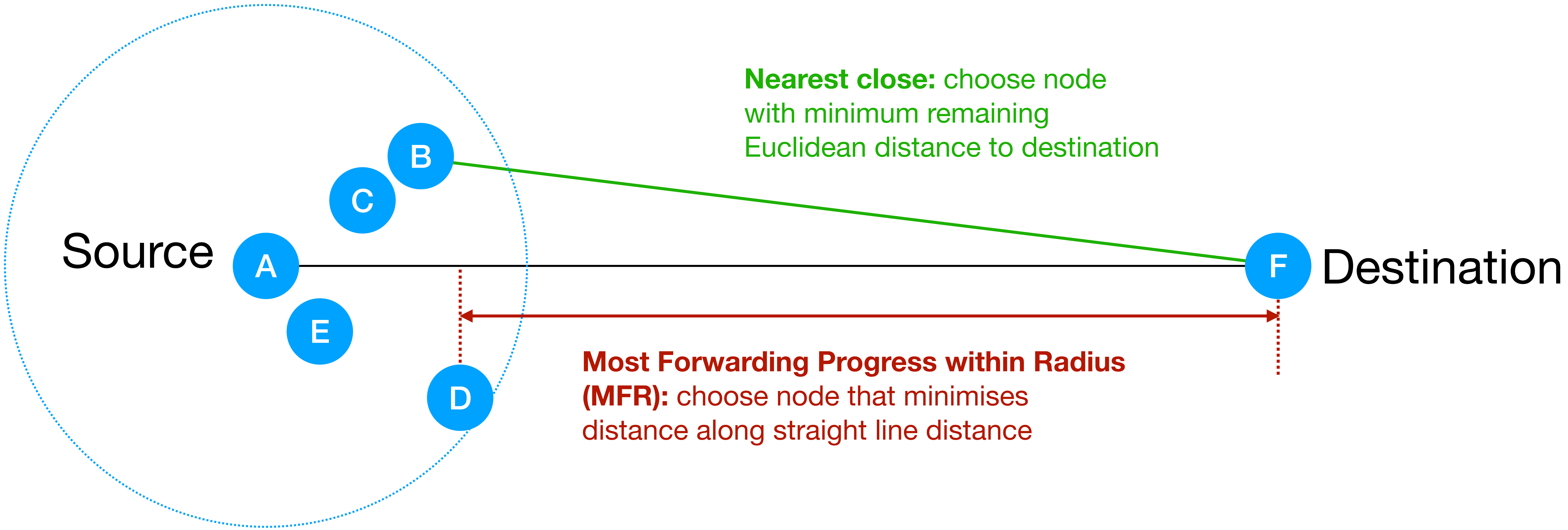
5

Geographic Routing: different distance metrics for Greedy Forwarding



- Different ways for measuring distances, and therefore, next hop to forward data to.

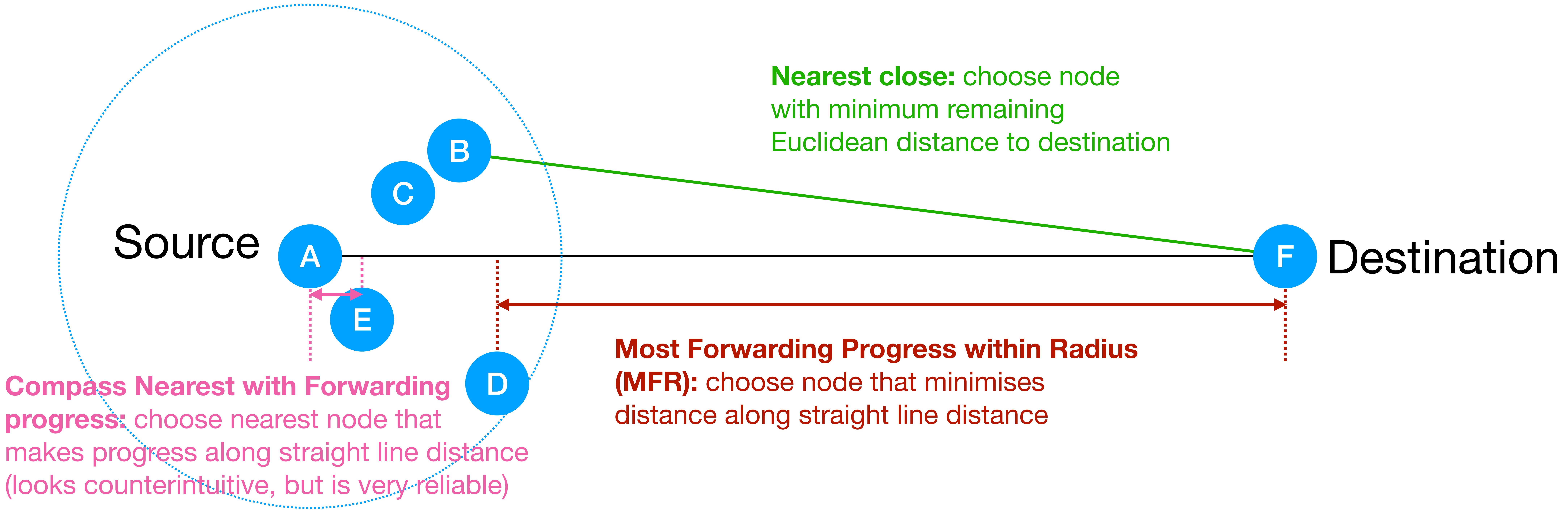
Geographic Routing: different distance metrics for Greedy Forwarding



- Different ways for measuring distances, and therefore, next hop to forward data to.

Geographic Routing: different distance metrics for Greedy Forwarding

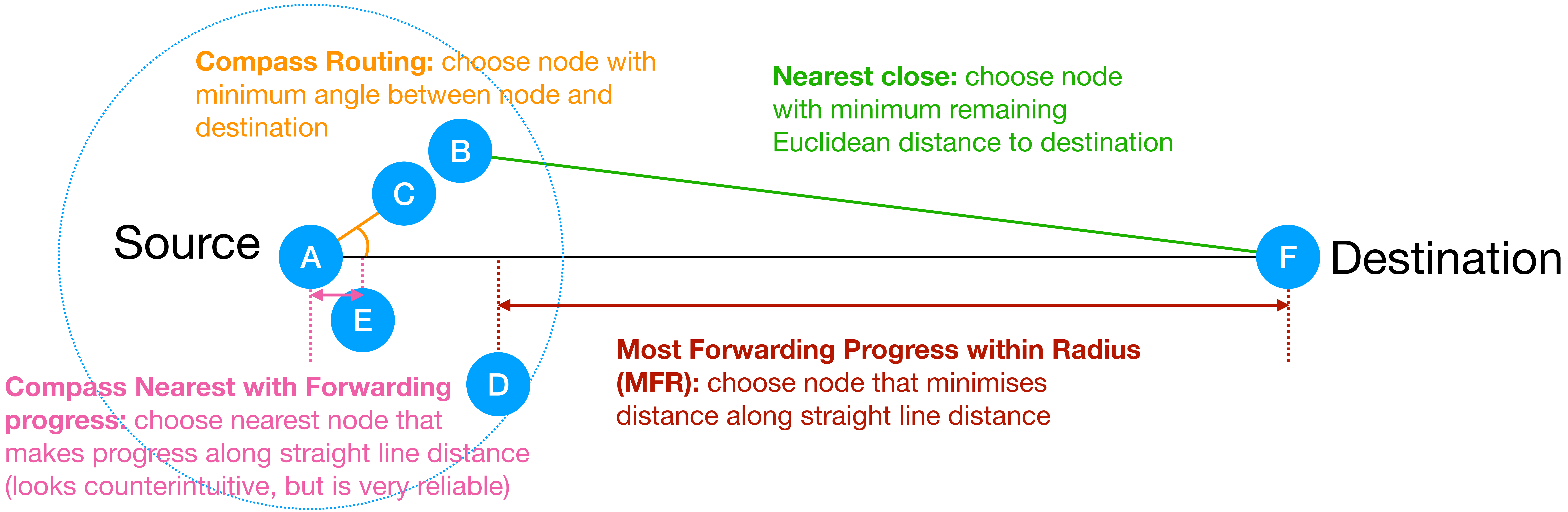
5



- Different ways for measuring distances, and therefore, next hop to forward data to.

5

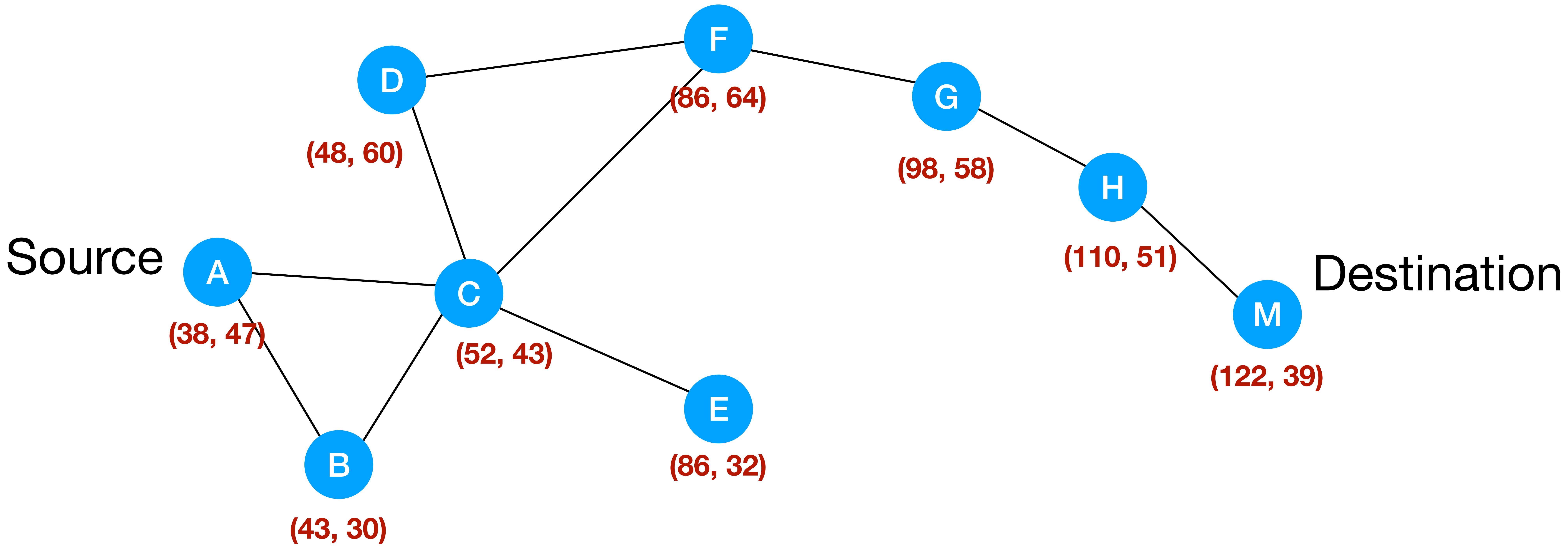
Geographic Routing: different distance metrics for Greedy Forwarding



- Different ways for measuring distances, and therefore, next hop to forward data to.

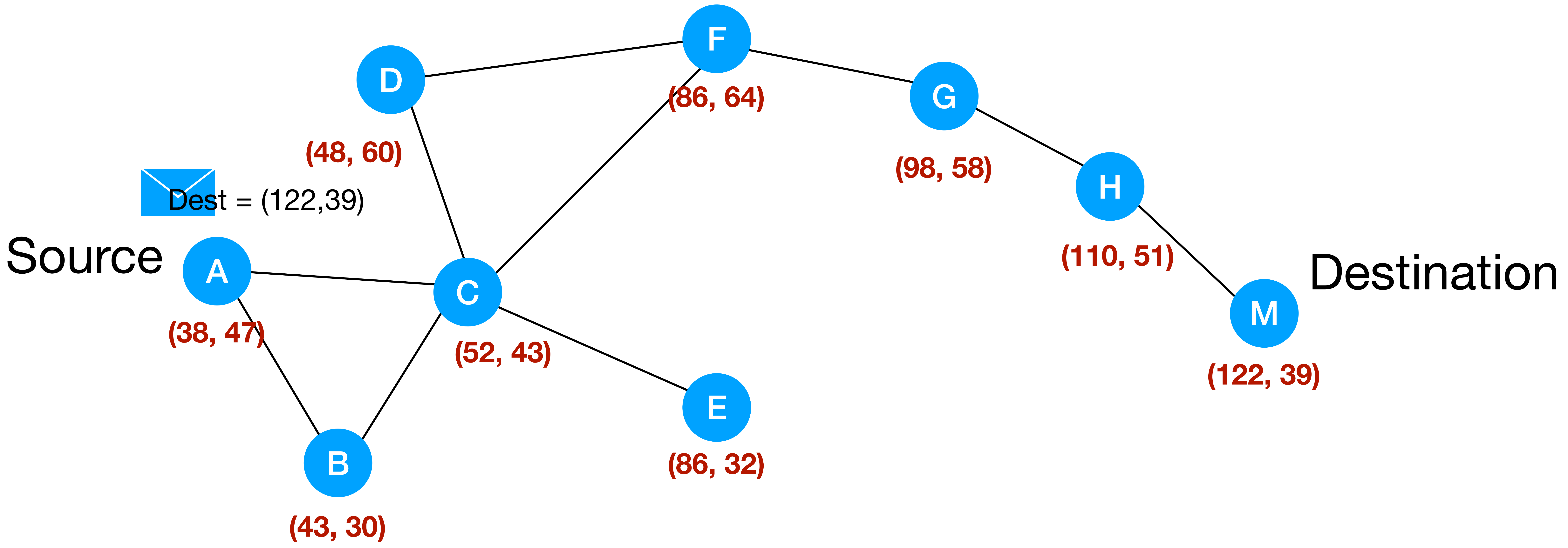
Geographic Routing: Greedy routing does not guarantee delivery

5



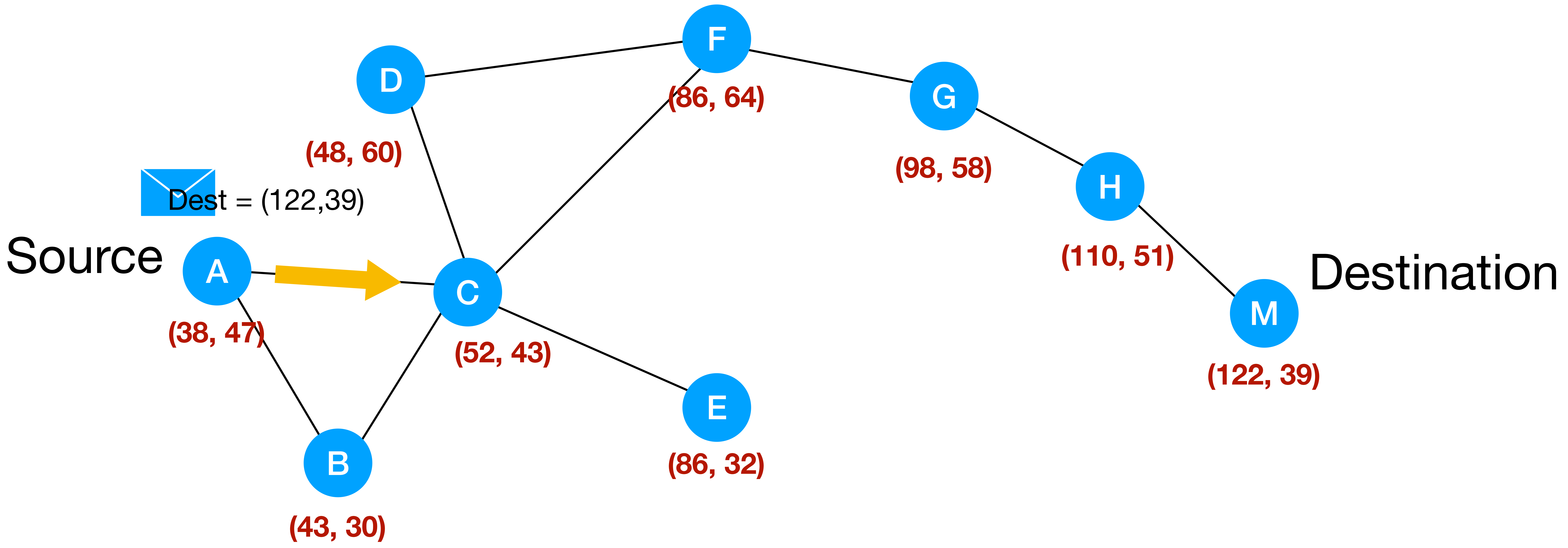
- Problem: Voids

Geographic Routing: Greedy routing does not guarantee delivery



- Problem: Voids

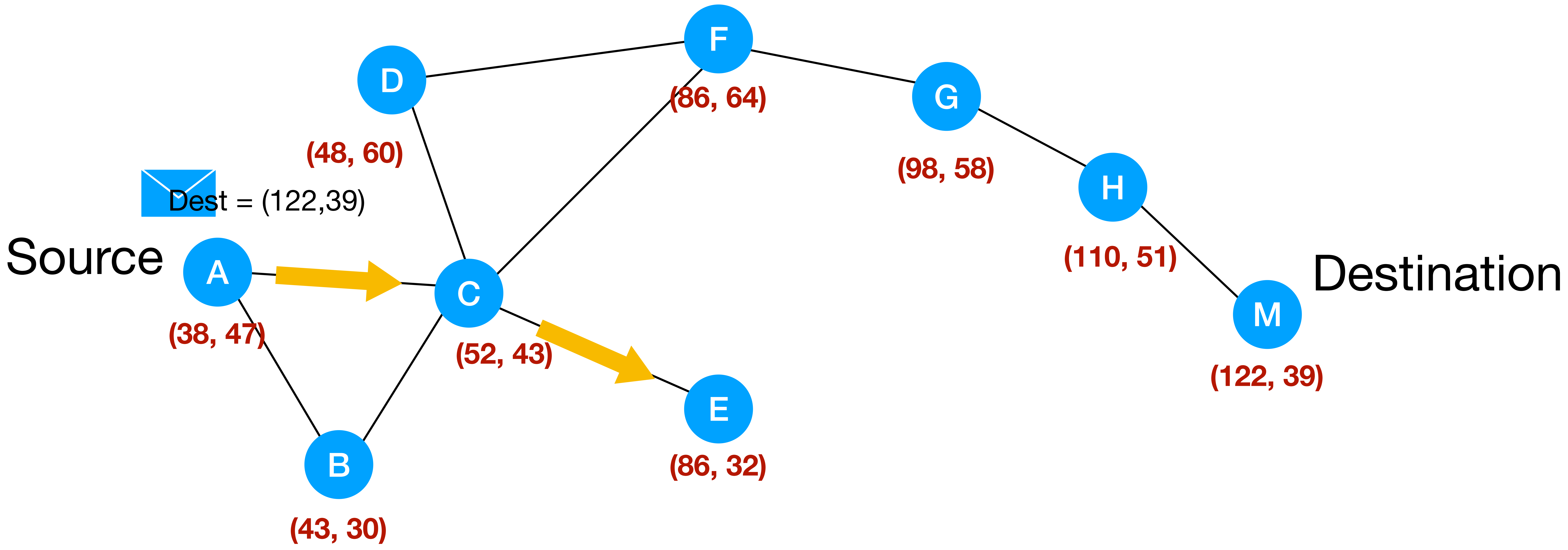
Geographic Routing: Greedy routing does not guarantee delivery



- Problem: Voids

5

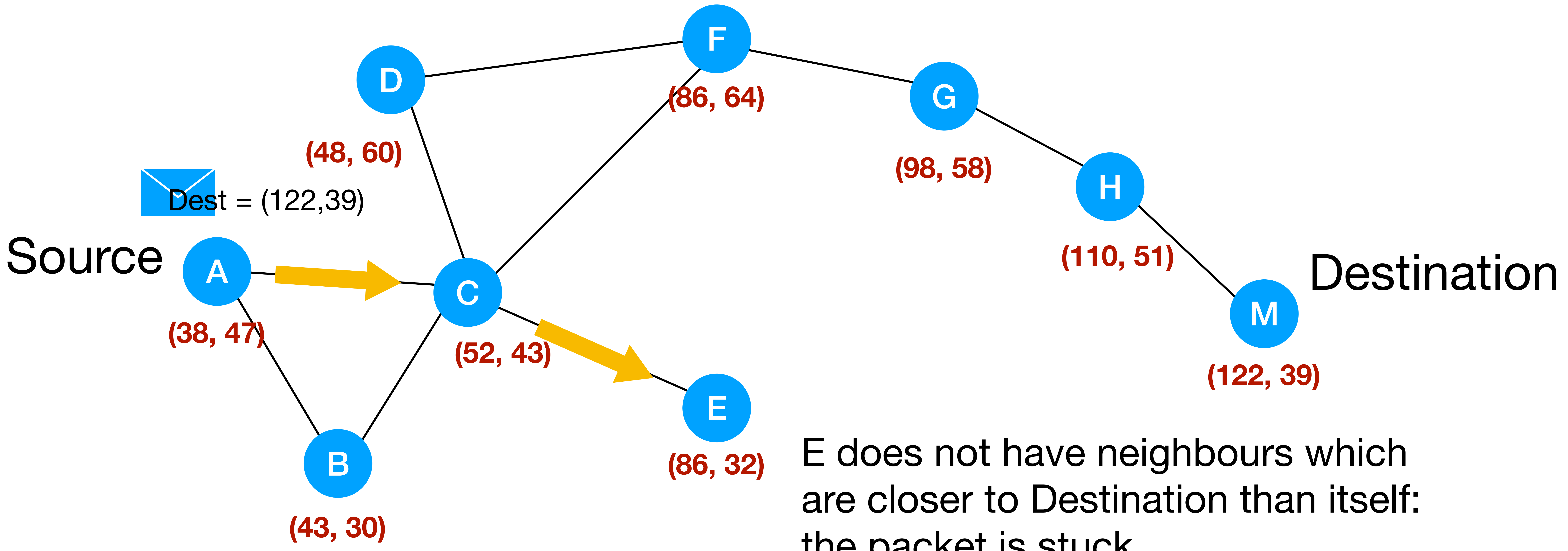
Geographic Routing: Greedy routing does not guarantee delivery



- Problem: Voids

5

Geographic Routing: Greedy routing does not guarantee delivery

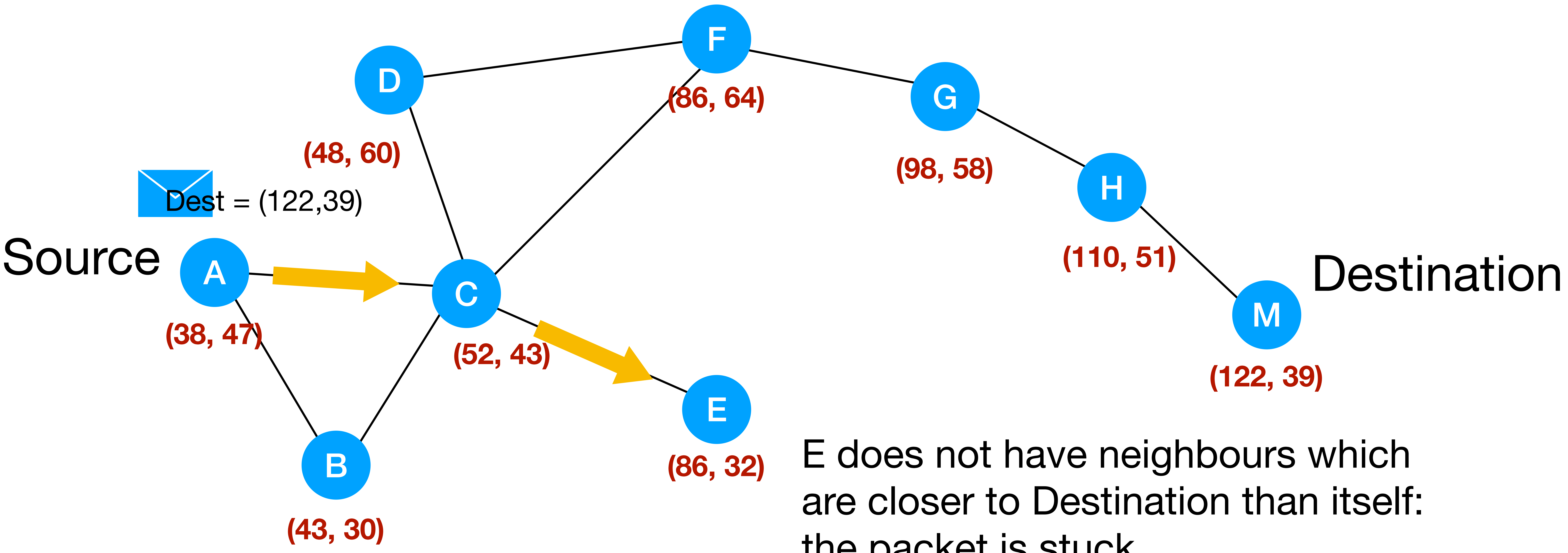


E does not have neighbours which are closer to Destination than itself: the packet is stuck

- Problem: Voids

5

Geographic Routing: Greedy routing does not guarantee delivery



E does not have neighbours which are closer to Destination than itself: the packet is stuck

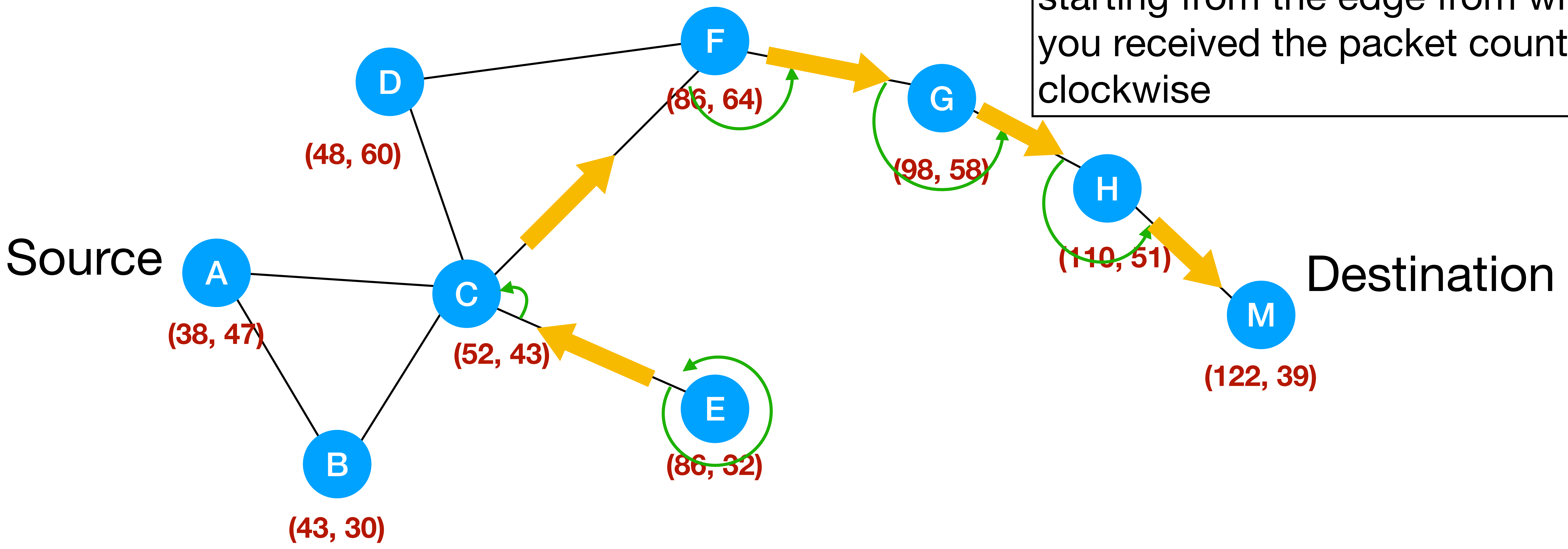
- Problem: Voids
- Solution: “Face routing”: switch face when we start going backwards

Geographic Routing: Greedy routing does not guarantee delivery

5

IDEA: use the right hand rule as when you walk in a maze.

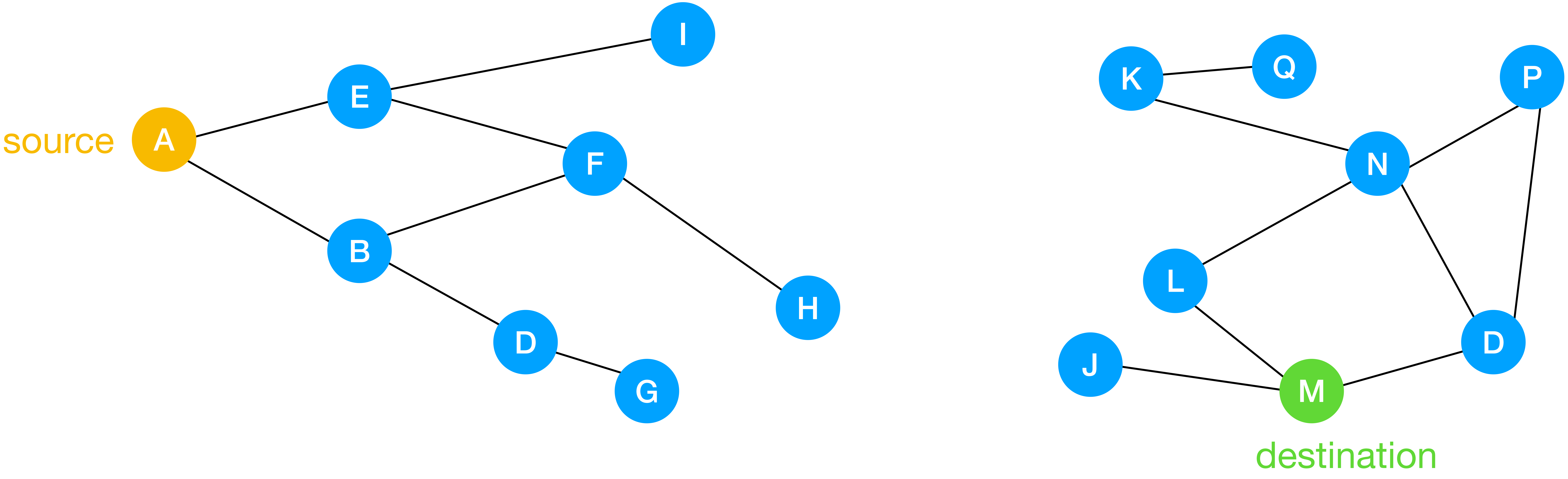
Send along the first edge you see starting from the edge from where you received the packet counter-clockwise



Back to our Motivation

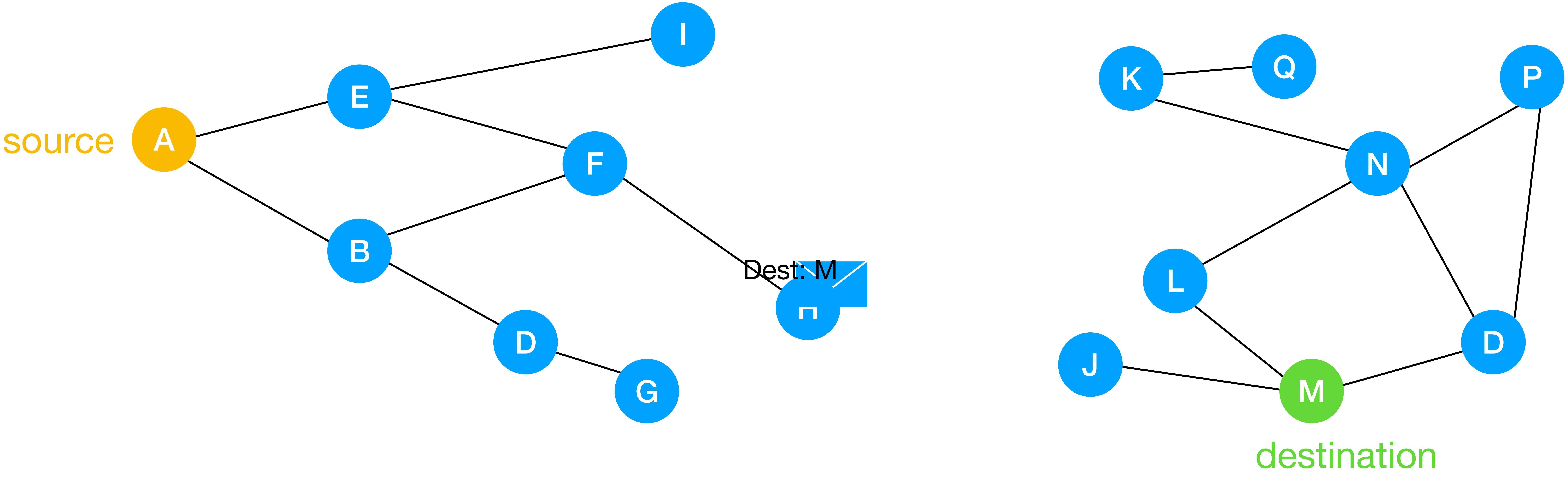
- We have been assuming that the network is not partitioned. This is not always realistic in IoT.
  - **occasionally connected networks:** sensors mounted on animals, floating in sea, space satellites that "pass" occasionally.
  - **Highly unreliable environments:** military networks suffering from jamming, acoustic links in air/water, free-space optical communications
  - **Low-power environments:** low-duty cycle sensors/actuators.

Delay-Tolerant Networking



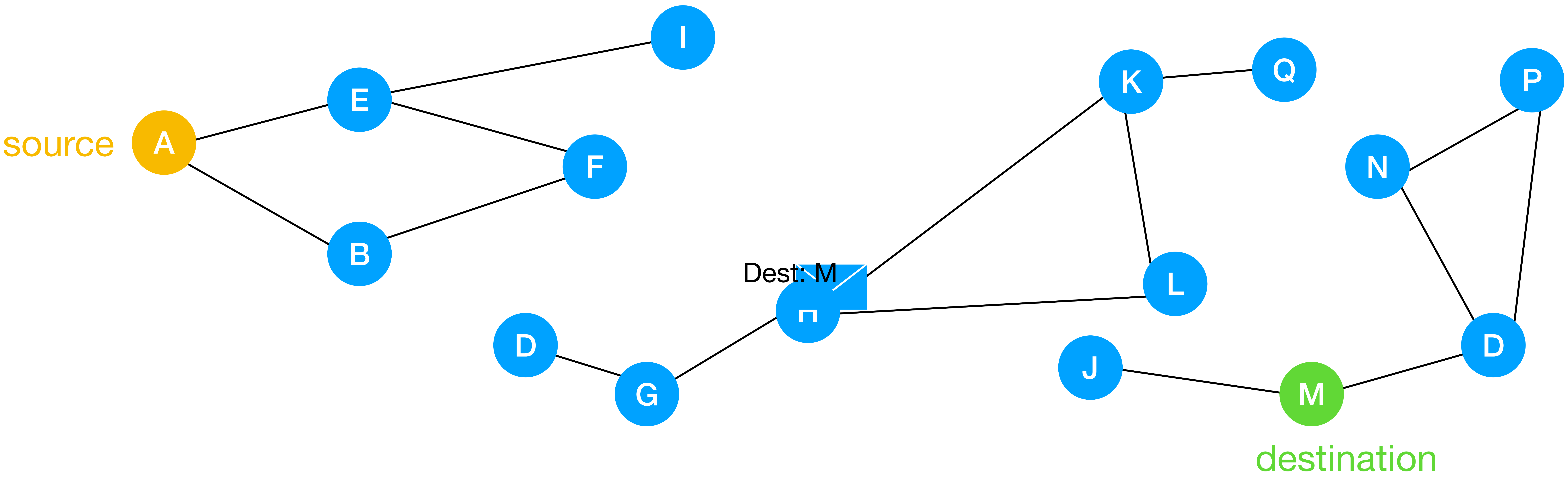
- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

Delay-Tolerant Networking



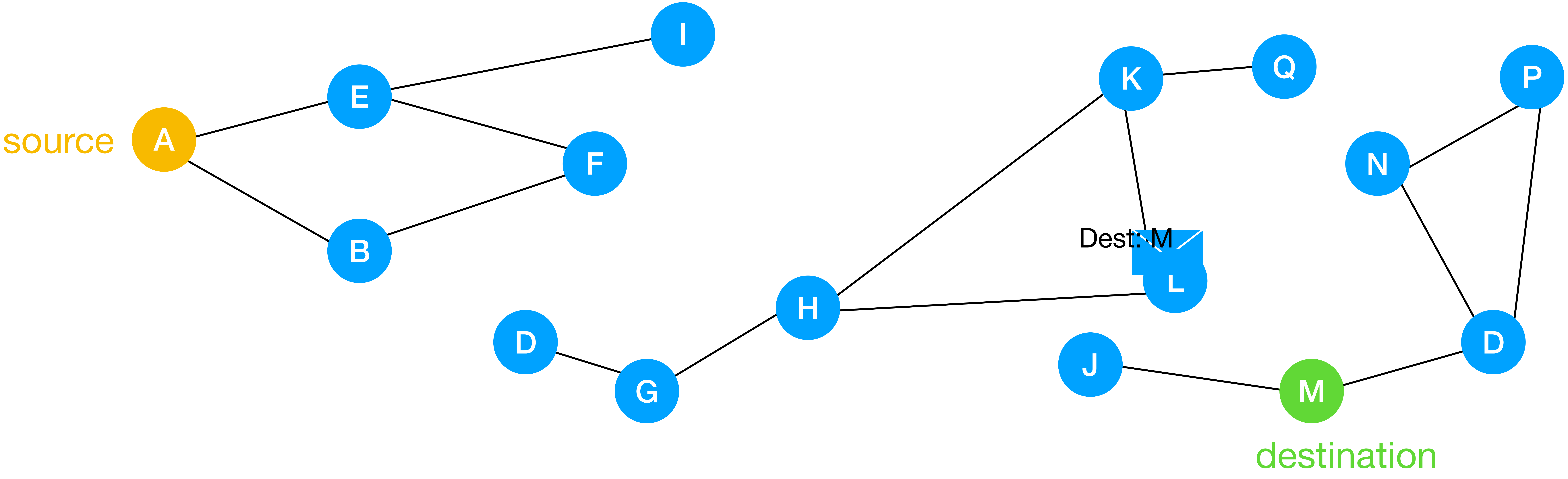
- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

Delay-Tolerant Networking



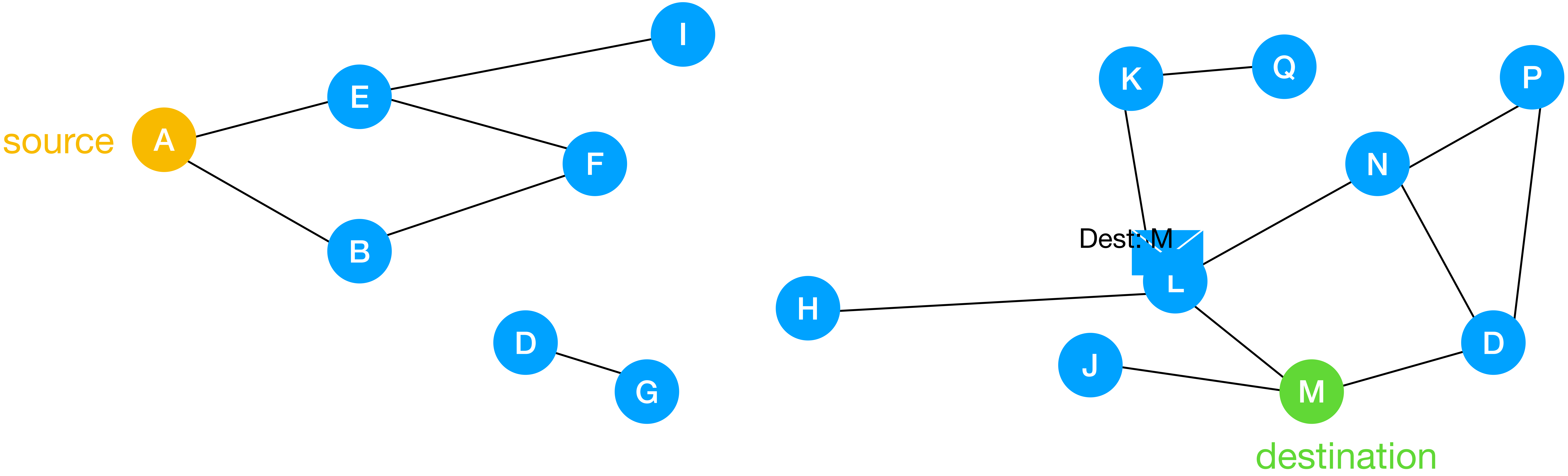
- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

Delay-Tolerant Networking



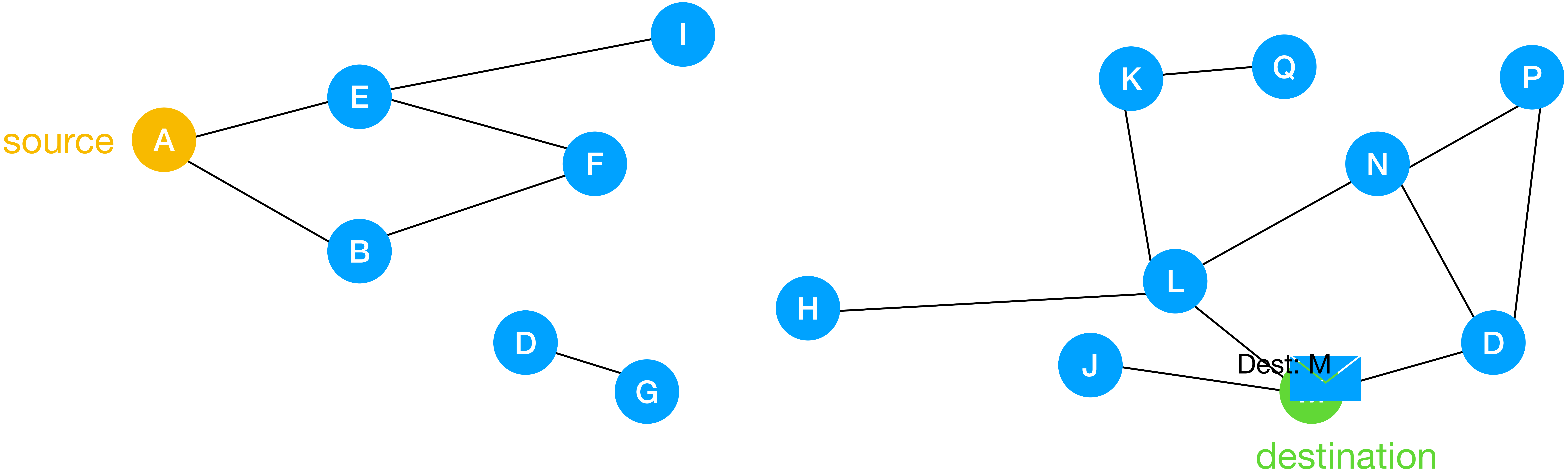
- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

Delay-Tolerant Networking



- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

Delay-Tolerant Networking



- Forward data as much as possible.
- Wait until the network topology changes to send the packet until it reaches destination.

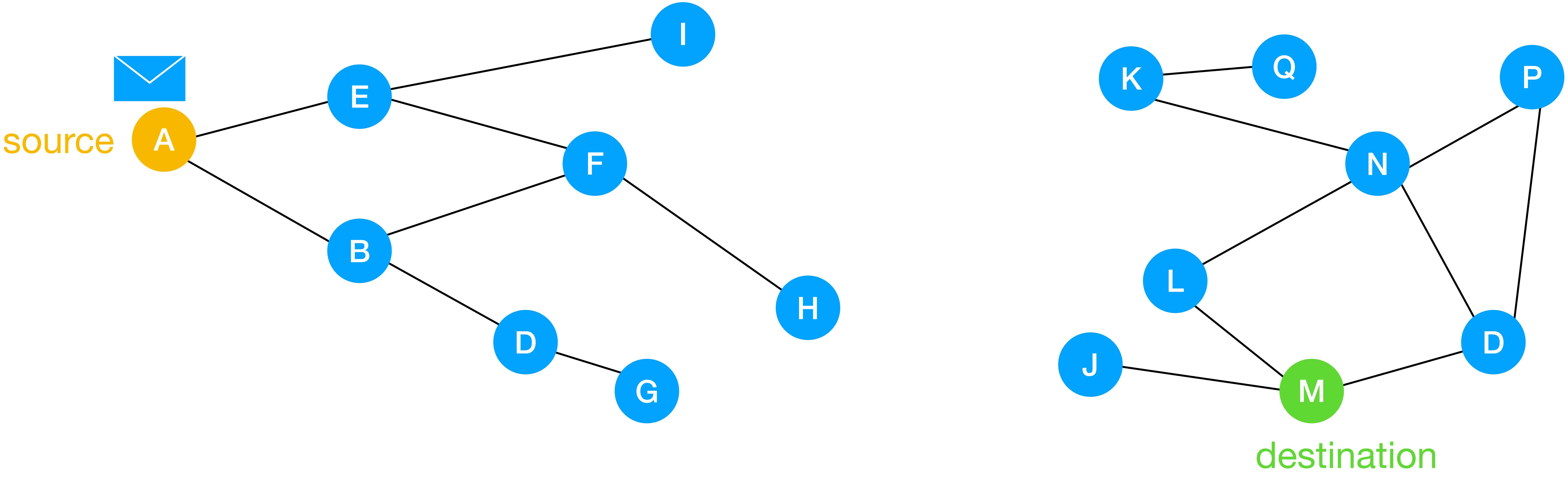
6

Delay-Tolerant networking: Gossip Algorithms (1)

- Idea: there is new rumor, people start gossiping about that. Initially they gossip a lot and the rumor spreads out very quickly, then people get bored about the rumor and mention it more rarely.

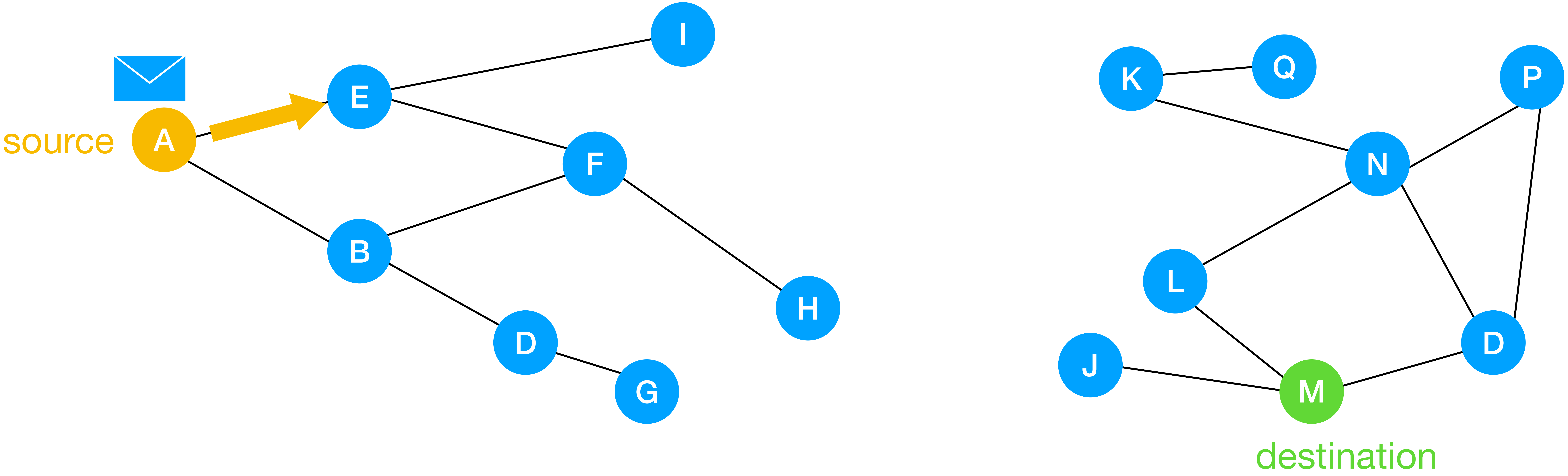
- If a node has data to send, it waits a random amount of time, picks a random target direction and broadcasts to nodes towards that direction. Such nodes apply the same mechanism.

6 Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

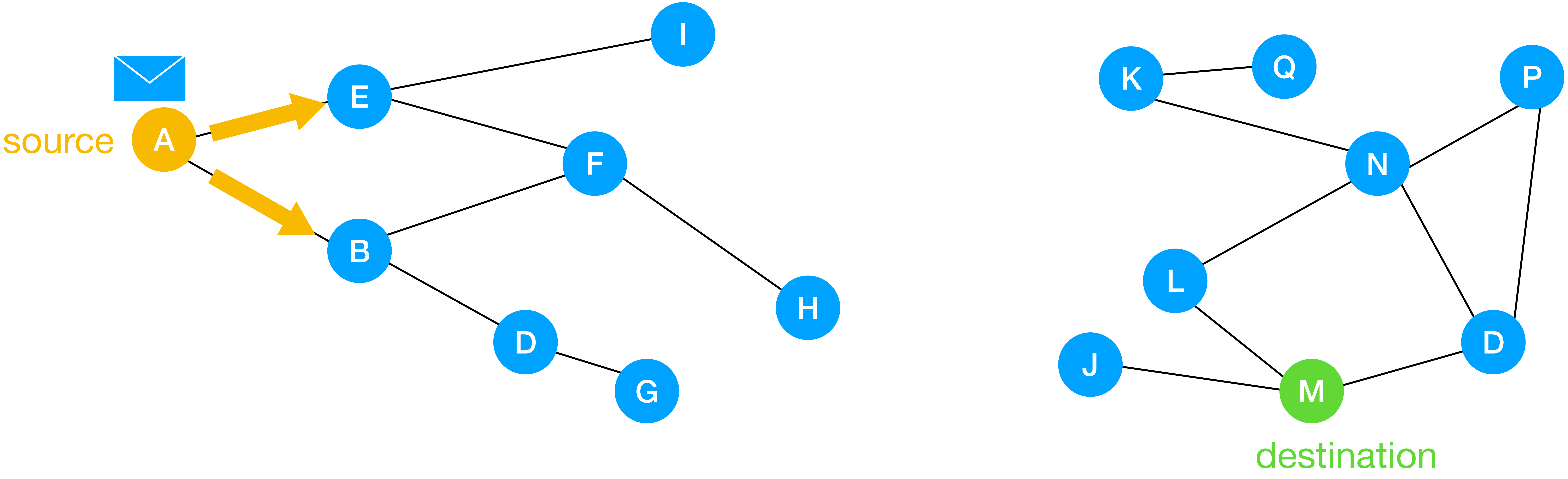
6 Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

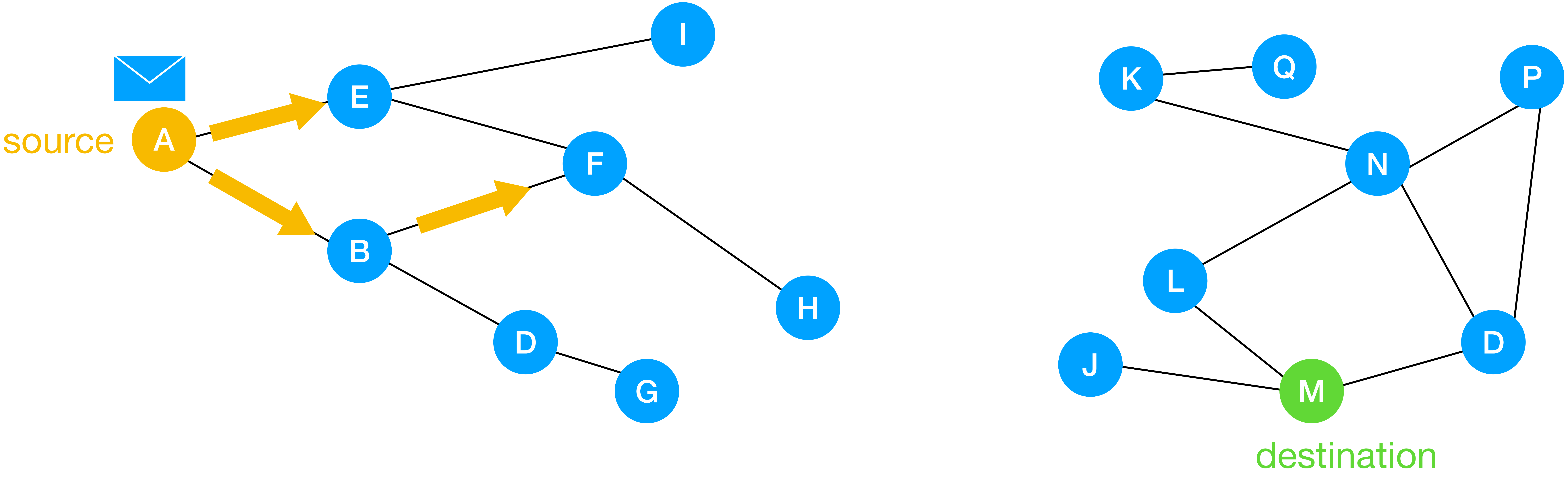
6

Delay-Tolerant networking: Gossip Algorithms (2)



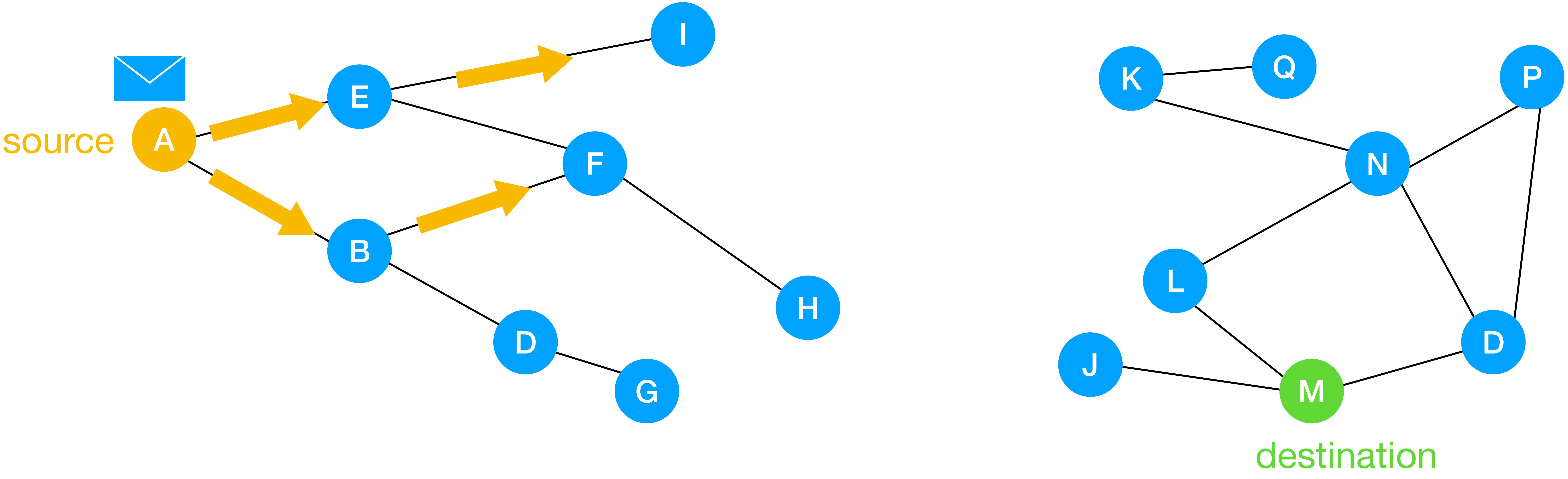
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



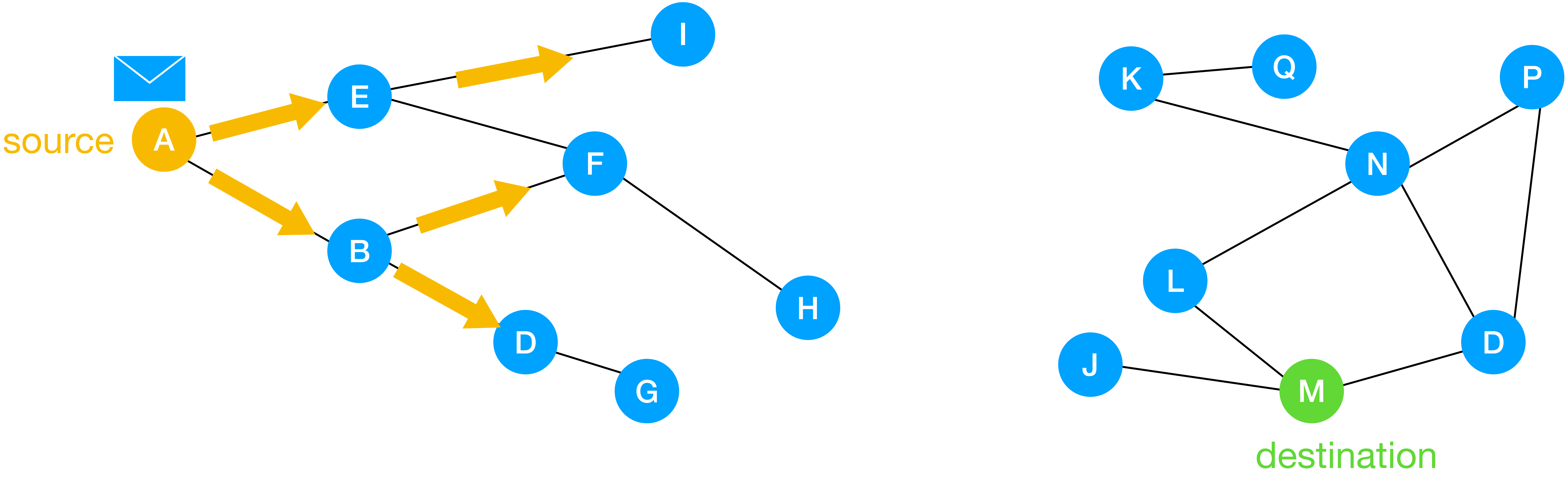
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



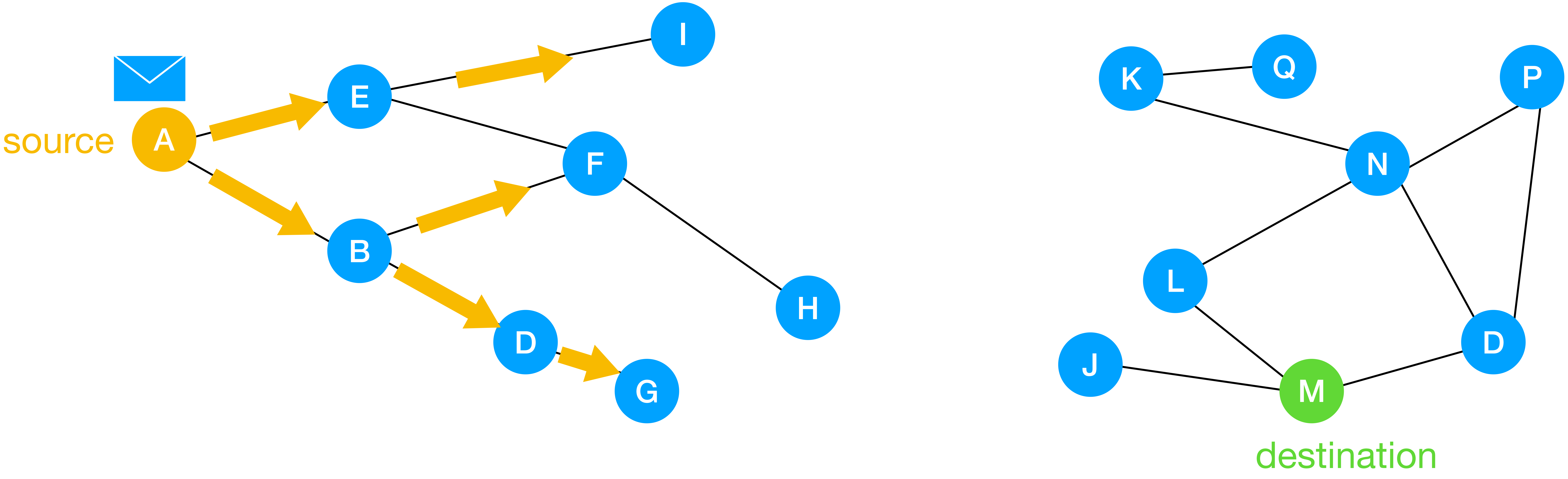
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



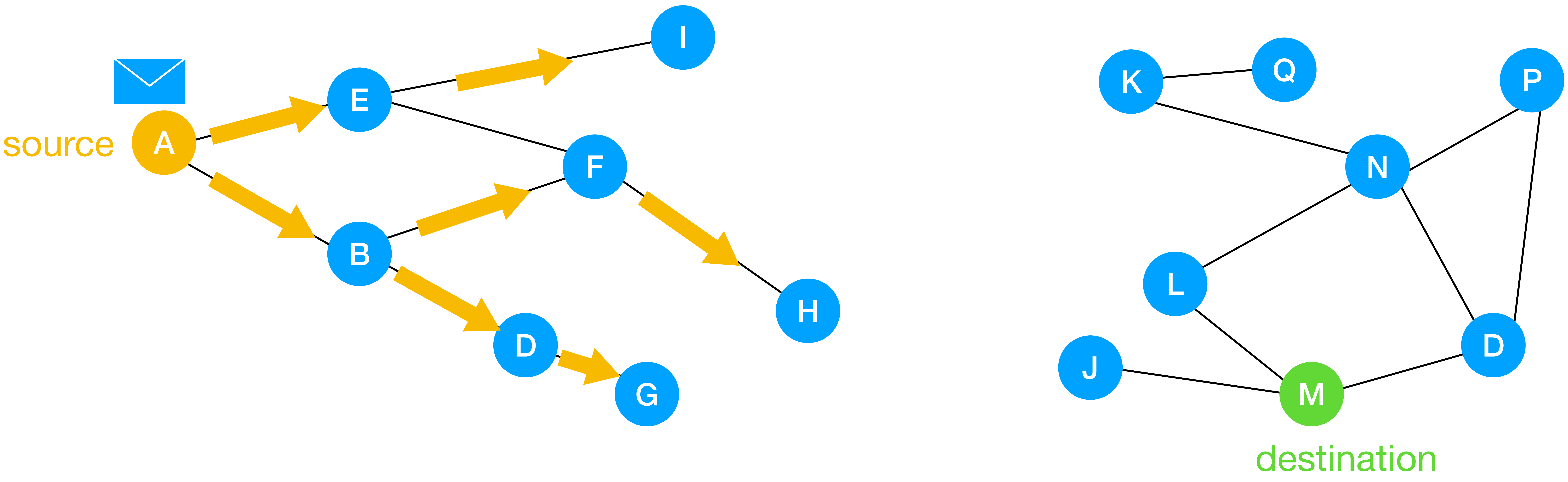
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



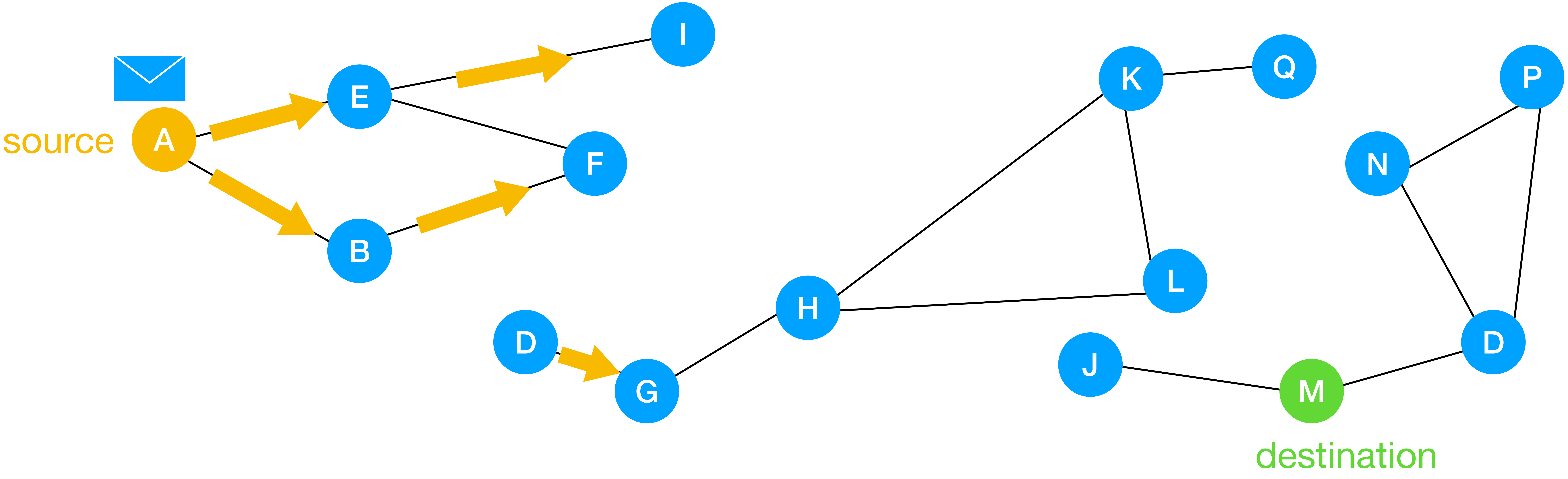
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



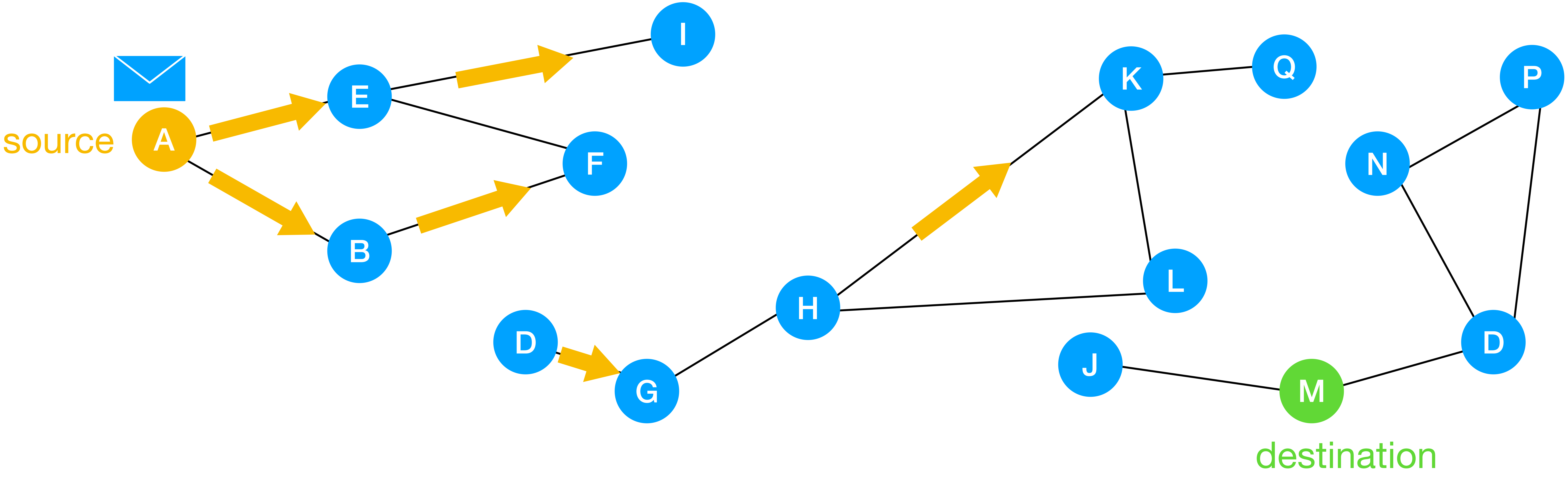
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



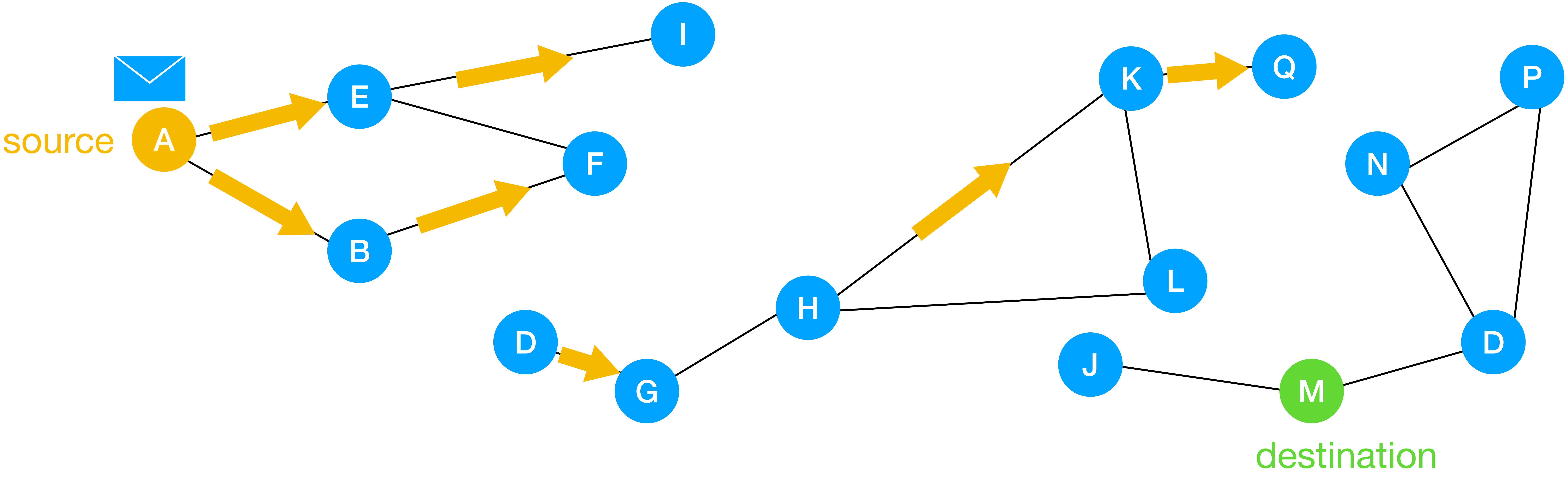
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



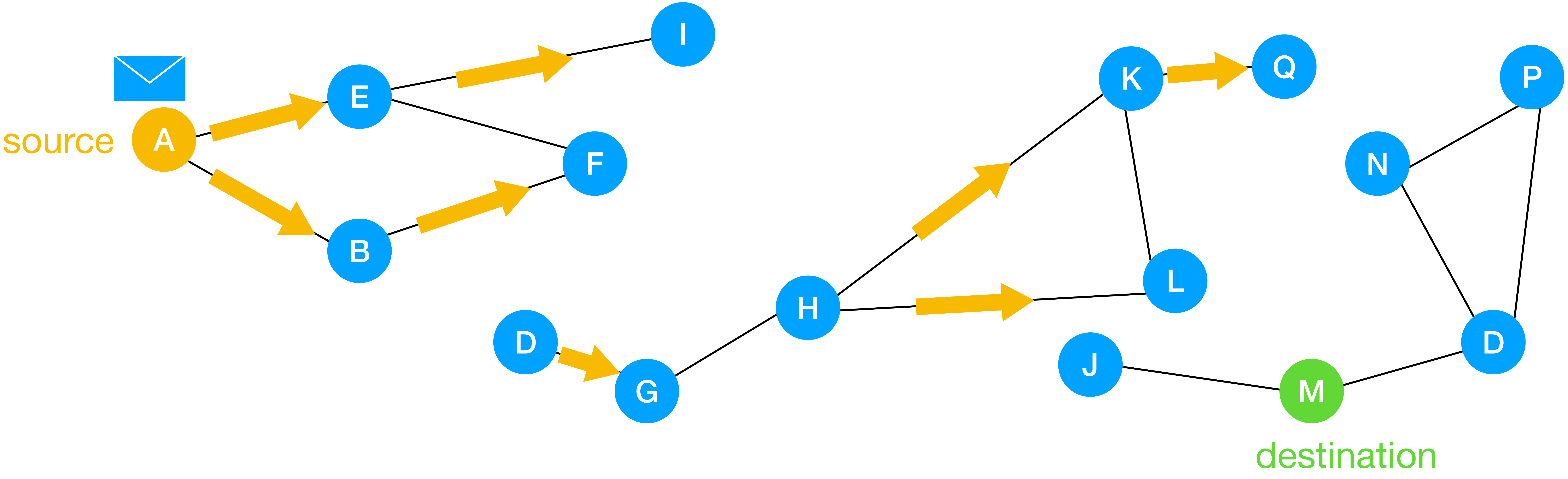
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



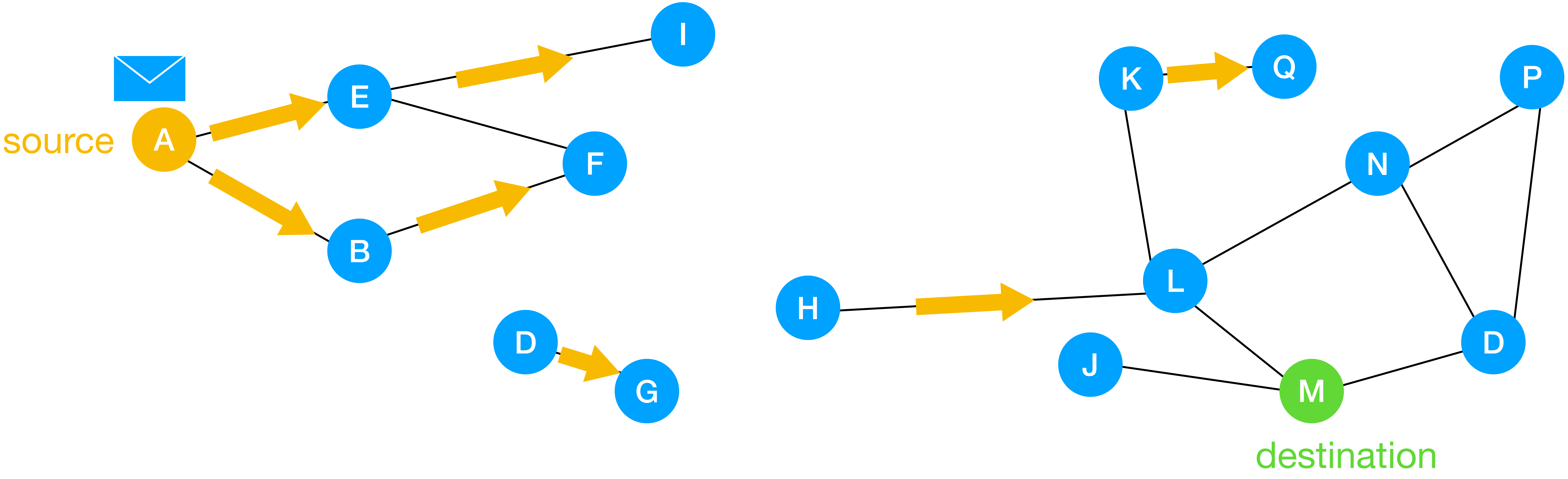
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



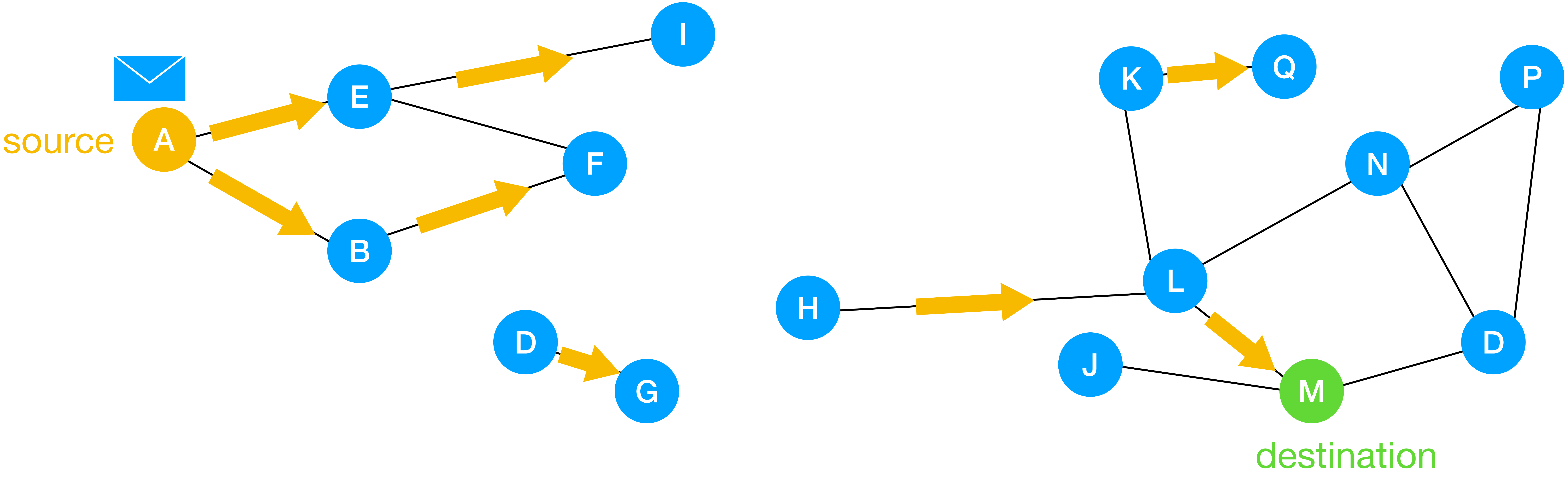
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



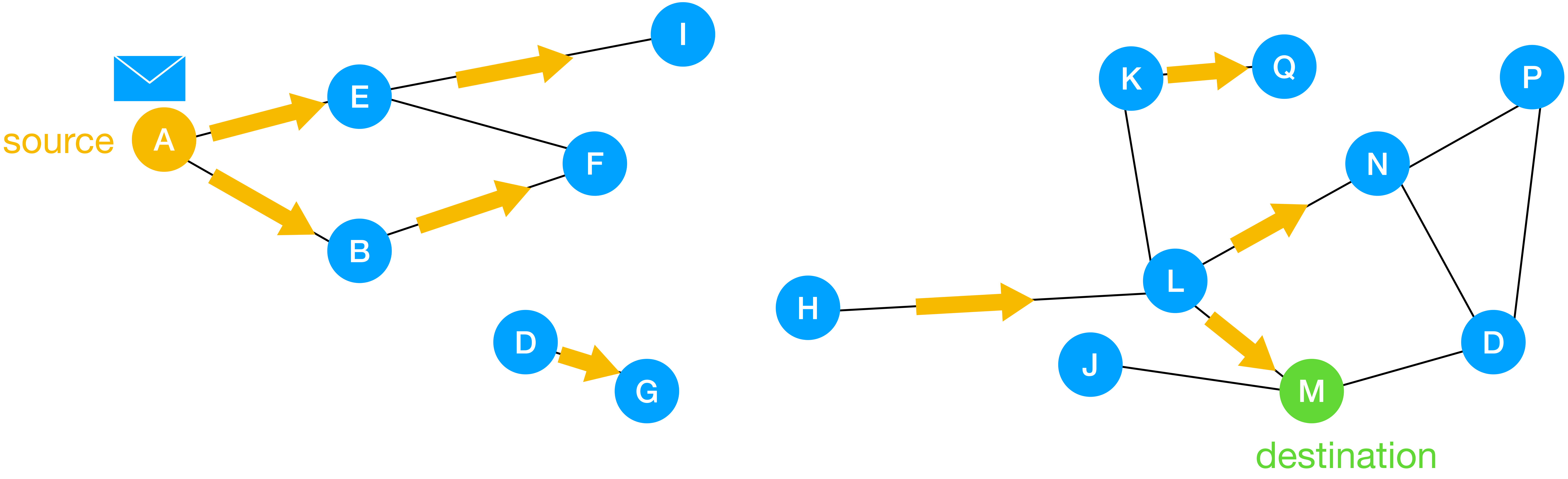
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



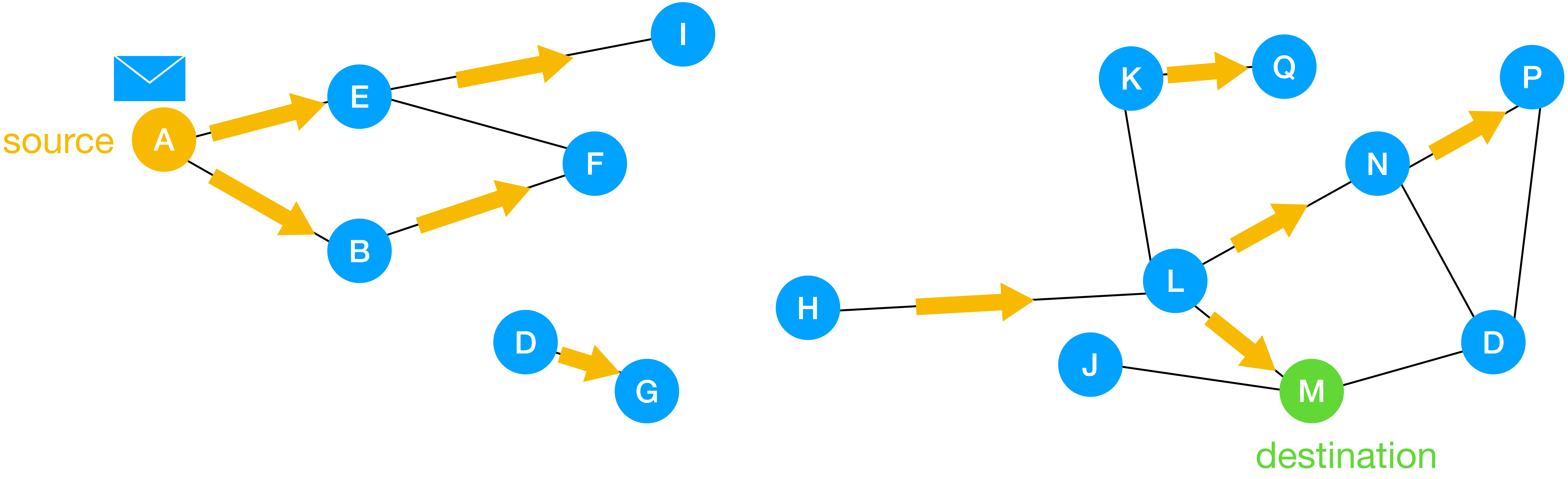
- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6 Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

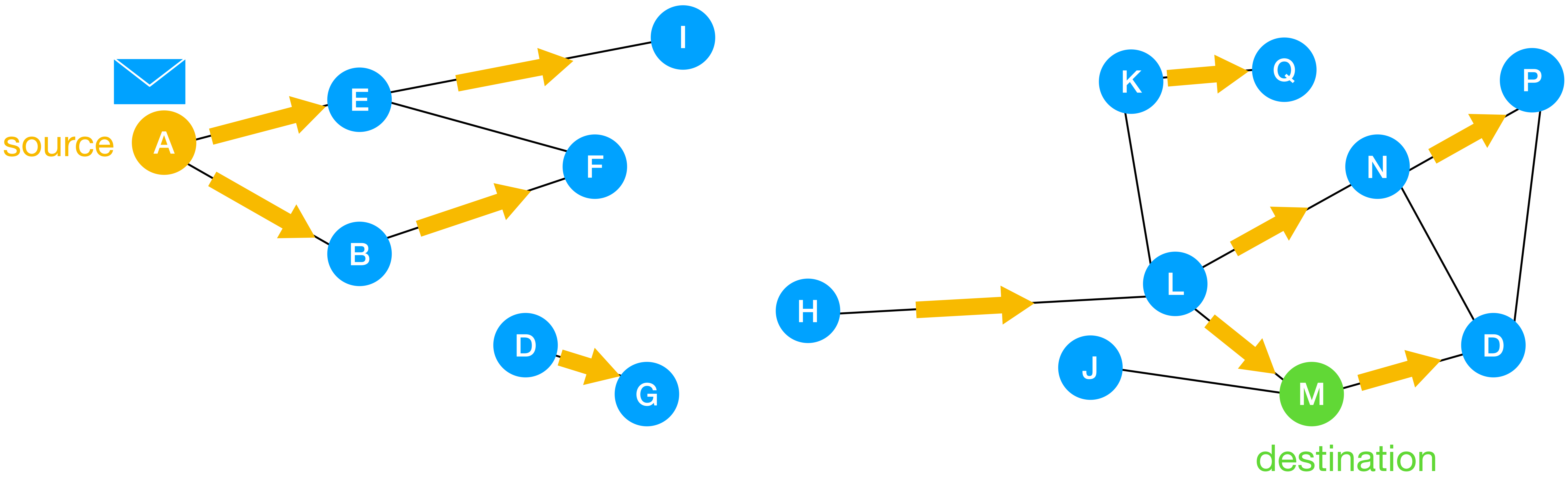
6 Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6

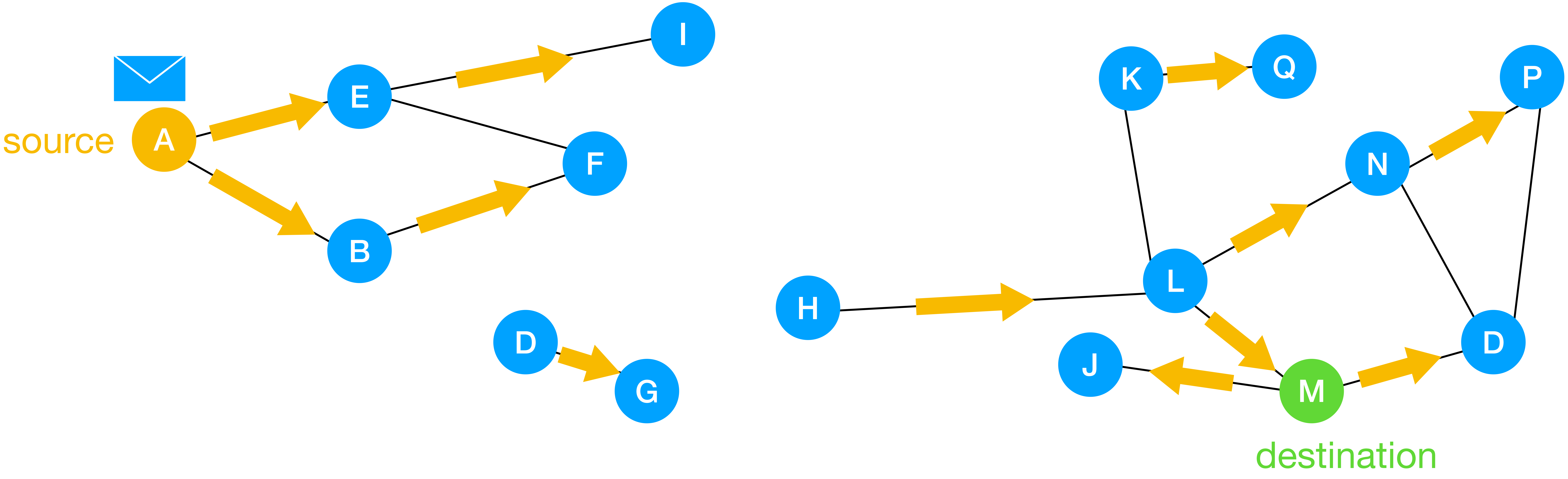
Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6

Delay-Tolerant networking: Gossip Algorithms (2)



- Every node waits random time after it receives data, picks random targets and sends data
- Some nodes might receive the same data multiple times and can decide to send it out again

6

Delay-Tolerant networking: Gossip Algorithms (3)

- If all link failures are transient and reoccurring, message will eventually reach the destination.
- Very simple algorithm, easy to implement, will likely reach all other nodes too (good for broadcasting).
- Slow propagation.
- Variant: **Rumor mongering**
  - When nodes get new update, it becomes a “hot rumor” (probability of sending out packets is higher)
  - When a node hears the packet many times, it propagates it less frequently.

Bibliography

- IoT Communication, University of Illinois Urbana-Champaign
- https://www.rfc-editor.org/rfc/rfc3626.html
- https://datatracker.ietf.org/doc/html/rfc4728
- https://www.rfc-editor.org/rfc/rfc3561
- Ha, Jae Yeol, et al. "EHRP: Enhanced hierarchical routing protocol for zigbee mesh networks ee." IEEE communications letters 11.12 (2007): 1028-1030..
- I. Stojmenovic, "Position-based routing in ad hoc networks," in IEEE Communications Magazine, vol. 40, no. 7, pp. 128-134, July 2002, doi: 10.1109/MCOM.2002.1018018.