Lesson 4.3: **Network Layer**

CSC450 – COMPUTER NETWORKS | WINTER 2019-20

DR. ANDREY TIMOFEYEV

OUTLINE

- Routing algorithms.
  - Introduction.
  - Routing algorithms classification.
  - Link-state routing.
  - Distance-vector routing.
  - Link-state vs. distance-vector.
  - Hierarchical routing.
  - Open Shortest Path First (OSPF) protocol.
  - Broadcast routing.

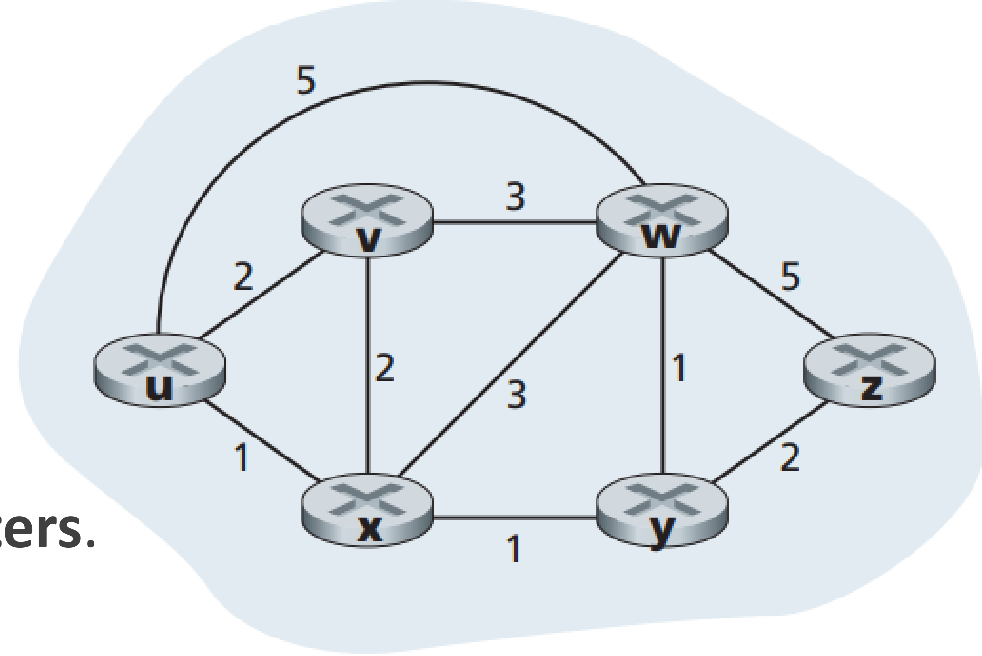
INTRODUCTION (1)

- Network layer **routing function** – determine the **route** taken by **packets** from **source** to **destination**.
  - **Host** is attached **directly** to **one router** – **default** or **first-hop** router.

- **Goal** of **routing protocols** – determine “***good***” **paths** (*routes*) from **sending** host to **receiving** host through network of **routers**.
  - **Path** (*route*) – sequence of routers that **packet** will **traverse** while going from given **source default router** to **destination default router**.
  - “***Good***” **path** – least-cost, fastest, or least congested.

INTRODUCTION (2)

- A network is **abstracted** as a **graph** to formulate **routing problems**.
  - Graph ***G = (N, E)***
  - **N** = set of **routers** (**nodes**) = *{ u, v, w, x, y, z }*.
  - **E** = set of **links** (**edges**) = *{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }*
- Every **link** has a **cost** associated with it.
  - **c(u,v)** = cost of link between node *u* and *v* – *(u, v)*.
    - *c(u, w) = 5*
    - *c(u, z) =* ∞ (no direct link)
    - *c(u, v) = c(v, u)* – undirected graph
  - **Cost** of **path** $(x_1, \ldots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p)$.
- **Routing algorithms** find the **least-cost path** between **routers**.



Abstract graph model of a computer network

ROUTING ALGORITHMS CLASSIFICATION

- Two ways of **classifying routing algorithms**:
  - **Global** vs. **decentralized** algorithms.
    - **Global:**
      - All routers have complete network topology and all links cost information.
      - **Link-state routing.**
    - **Decentralized:**
      - Router knows only physically-connected neighbors and link costs to those neighbors.
      - Iterative process of computation and exchange of information with neighbors.
      - **Distance-vector routing.**
  - **Static** vs. **dynamic** algorithms.
    - **Static:**
      - Routes change slowly over time.
    - **Dynamic:**
      - Routes change more frequent.
        - In response to link cost change.
        - Periodic update.

LINK-STATE ROUTING

- **Link-state (LS) routing algorithms** use **network topology** and **all link costs** as an **input** to compute the **least-cost paths** in the network.
  - Each node **broadcasts link-state packets** to all other nodes.
    - **Link-state packets** contain information about **network nodes** and **all link costs**.
  - All nodes have **identical** and **complete views** of the **network**.
  - **Each node** runs **LS algorithm** to compute the **same set** of **least-cost paths**.

LINK-STATE ROUTING: DIJKSTRA’S ALGORITHM (1)

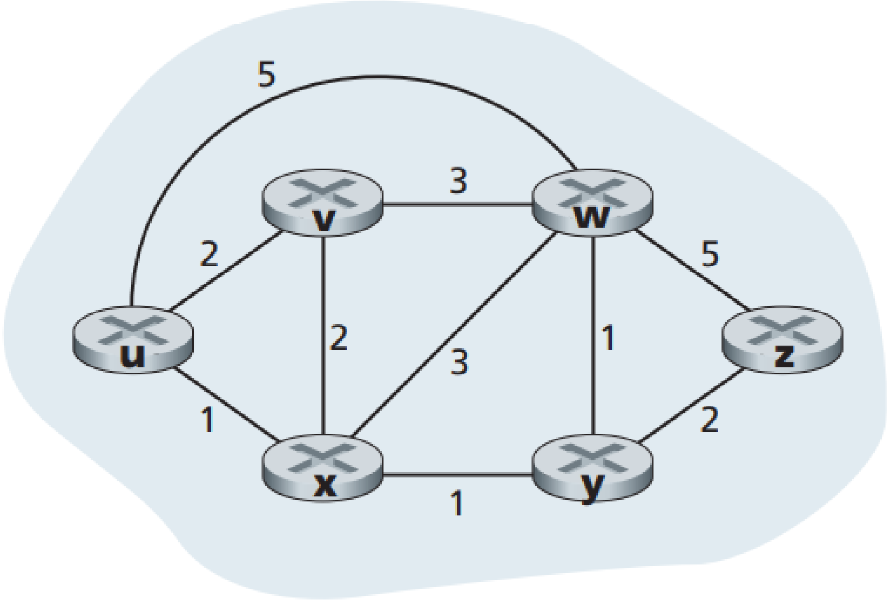
- **Dijkstra’s algorithm**:
  - Graph algorithm for computing the **least-cost path** from **one node** (“source”) to **all** other **nodes** in the **network**.
  - Provides **forwarding table** for **source node**.
- **Notations**:
  - ***D(v)*** – current cost of the least-cost path from source to destination v.
  - ***p(v)*** – previous node along the path from source to v.
  - ***N’*** – set of nodes whose least-cost path definitely known.

LINK-STATE ROUTING: DIJKSTRA'S ALGORITHM (2)

- **Dijkstra's algorithm** for **source** node ***u***:

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
      then D(v) = c(u, v)
    else D(v) = ∞

Repeat
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N':
    D(v) = min(D(v), D(w) + c(w, v))
    /* new cost to v is either old cost to v or known
       shortest path cost to w plus cost from w to v */
until all nodes in N'
```

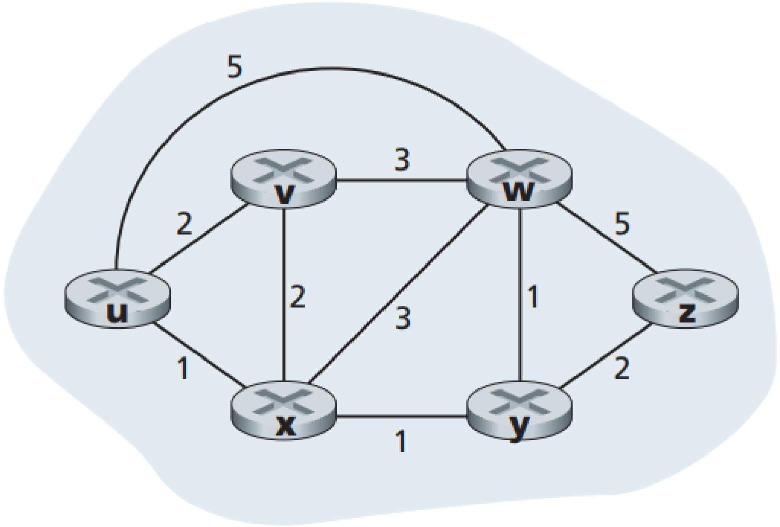


Network graph

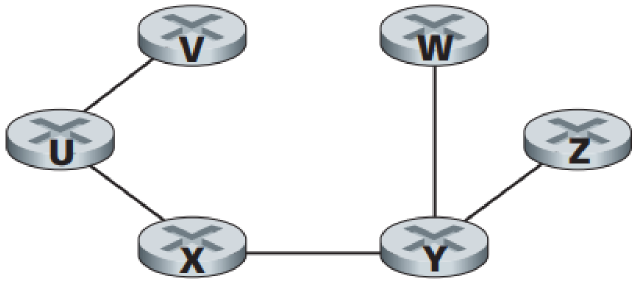
LINK-STATE ROUTING: DIJKSTRA'S ALGORITHM (3)

- **Dijkstra's algorithm** constructs **shortest path tree** by **tracing** the **predecessor nodes**.
  - **Forwarding table** for source node is constructed by storing the **next-hop node** on the **least-cost path**.

Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					



- <u>**Exercise:**</u>
  - Apply Dijkstra's algorithm for source node *z*.
  - Show trace table of the algorithm.
  - Show resulting shortest path tree (for node *z*).
  - Show resulting forwarding table (for node *z*).



Shortest path tree

Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Forwarding table

LINK-STATE ROUTING: DIJKSTRA'S ALGORITHM (4)

- **Dijkstra's algorithm complexity**:
  - *n* destination nodes, **not counting** the **source node**.
  - **Each iteration**: need to check **all** nodes not in N' with **minimum cost**.
  - **n(n+1)/2** comparisons $\rightarrow$ **O(n^2) complexity**.
    - More **efficient** implementation possible: **O(n*logn) complexity**.

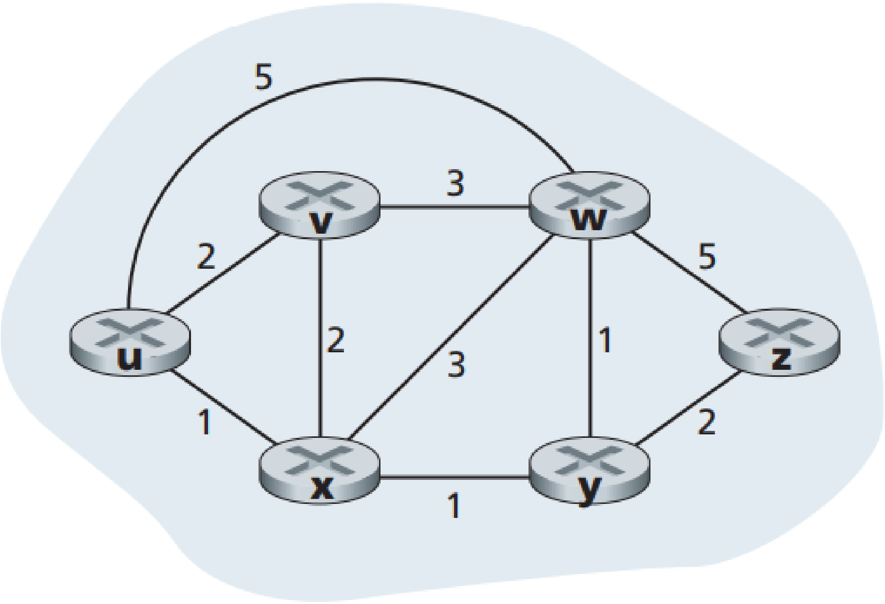
DISTANCE-VECTOR ROUTING (1)

- In the **distance-vector** (**DV**) **routing algorithm**, **node** only knows the **costs** of the links to its **directly attached neighbors** and the **costs** of these **neighbors' links**.
- **DV algorithm**:
  - **Distributed**.
    - Node receives information from direct neighbors, performs calculations, and distributes results back to neighbors.
  - **Iterative**.
    - Process continues until no more information is exchanged.
  - **Asynchronous**.
    - Does not require all nodes to operate in sync.

DISTANCE-VECTOR ROUTING (2)

- At a **core** of **DV algorithm** is **Bellman-Ford equation**:
  - ***$d_x(y) = \min_v\{c(x, v) + d_v(y)\}$***
    - ***$d_x(y)$*** – cost of least-cost path from source *x* to destination *y*.
    - ***$\min_v$*** – min function taken over all neighbors *v* of *x*.
    - ***$c(x, v)$*** – cost of link to neighbor *v*.
    - ***$d_v(y)$*** – cost from neighbor *v* to destination *y*.

- **Example:**
  - Cost of the least-cost path from *u* to *z*.
    - *u* has three neighbors: *v*, *x*, *w*.
      - $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$.
    - By Bellman-Ford equation:
      - $d_u(z) = \min\{c(u,v) + d_v(z),\ c(u,x) + d_x(z),\ c(u,w) + d_w(z)\}$
        $= \min\{2 + 5,\ 1 + 3,\ 5 + 3\} = 4$



Network graph

DISTANCE-VECTOR ROUTING (3)

- **Notations**:
  - $D_x(y)$ – estimate of cost of least-cost path from x to y
  - $D_x = [D_x(y): y \text{ in } N]$ – **distance vector** of cost estimates from *x* to all other nodes in *N*.
- In **DV algorithm** each **node** *x* maintains:
  - $c(x, v)$ – **cost** to each attached neighbor.
  - $D_x = [D_x(y): y \text{ in } N]$ – **own** distance vector.
  - $D_v = [D_v(y): y \text{ in } N]$ – distance vectors of each of its **neighbors**.
- **DV algorithm** description:
  - From time-to-time, **each node sends** its own **DV** estimate to **neighbors**.
  - When *x* **receives new DV** estimate from neighbor, it **updates** its own DV using **B-F equation**:
    - $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$ for each node *y* in N.
  - The **estimate $D_x(y)$ converges** to the actual **least-cost $d_x(y)$**.

DISTANCE-VECTOR ROUTING (4)

- **DV** is **iterative** & **asynchronous**.
  - Each local **iteration** caused by:
    - Local link cost **change**.
    - **DV update** message from neighbor.
- **DV** is **distributed**.
  - Each **node notifies neighbors** only when its DV **changes**.
    - **Neighbors** then **notify** their **neighbors** if necessary.
- **Each node:**

wait for [change in local link cost or msg from neighbor] → **re-compute** estimates → if DV to any dest has changed, **notify** neighbors

DISTANCE-VECTOR ROUTING (5)

- **DV algorithm** at each node *x*:

Initialization:

for all destinations y in N:

$D_x(y) = c(x, y)$ /* if y is not neighbor then $c(x, y) = \infty$ */

for each neighbor w

$D_w(y) = \infty$ for all destination y in N

for each neighbor w

send distance vector $D_x = [D_x(y)$: y in N] to w

Repeat

wait (until link cost change or DV received from some neighbor)

for each y in N:

$\mathbf{D_x(y) = min_v\{c(x, v) + D_v(y)\}}$

If $D_x(y)$ changed for any destination y

send DV $D_x = [D_x(y)$: y in N] to all neighbors

forever

DISTANCE-VECTOR ROUTING: EXAMPLE (1)

- **DV algorithm** at each node:

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

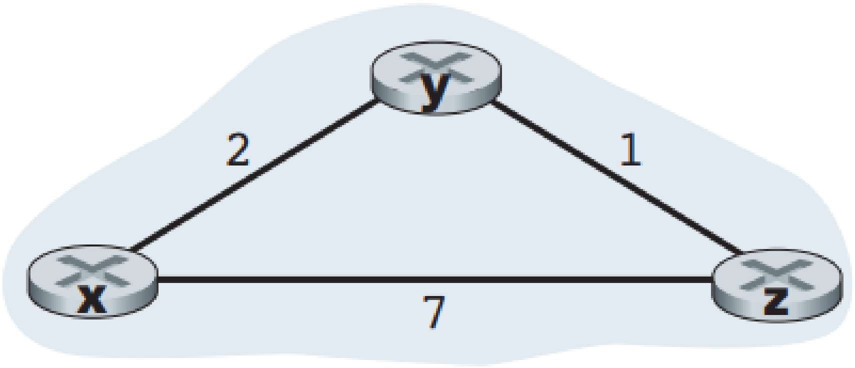
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



Network graph

time

DISTANCE-VECTOR ROUTING: EXAMPLE (2)

- **DV algorithm** at each node:

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node y table

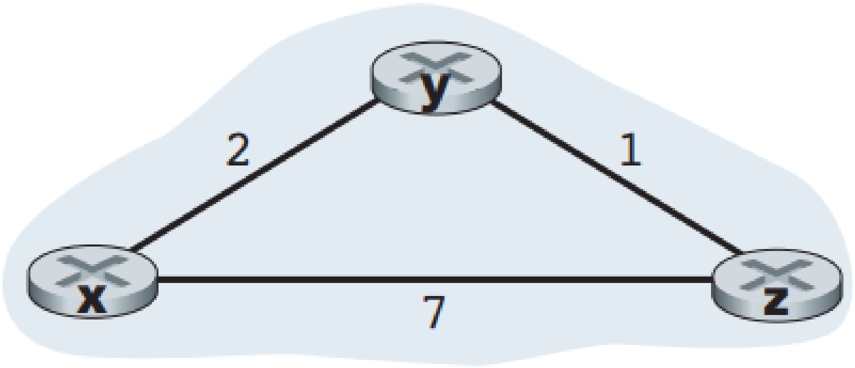
from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0



Network graph

time

DISTANCE-VECTOR ROUTING: EXAMPLE (3)

- **DV algorithm** at each node:

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

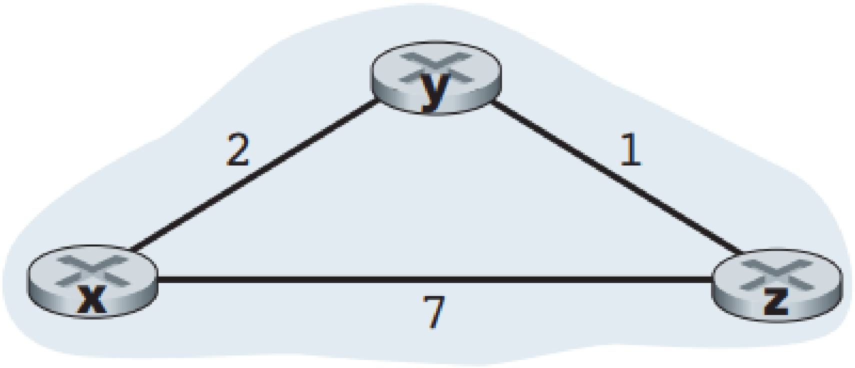
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



Network graph

time

LINK-STATE VS. DISTANCE-VECTOR

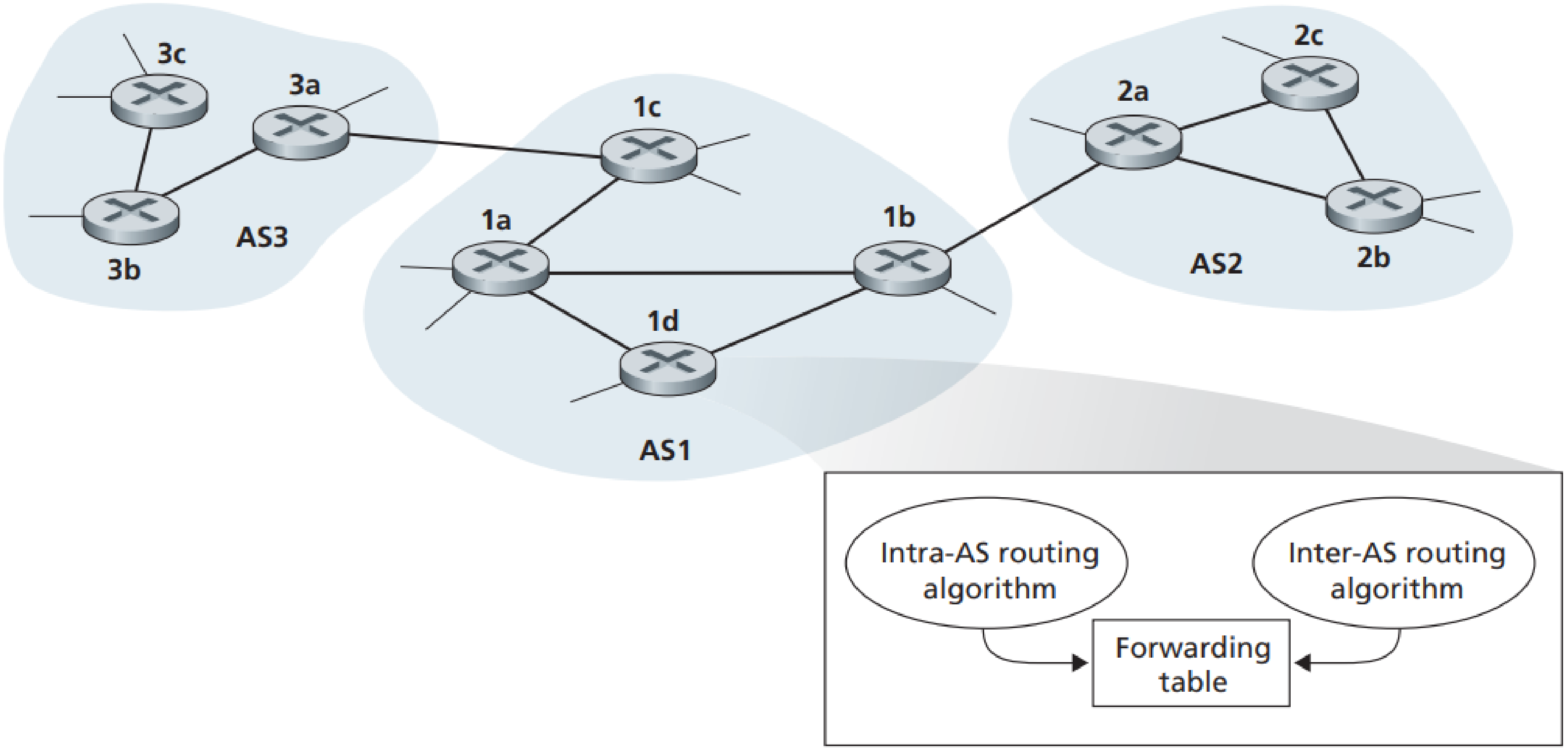
- **Comparison** between **link-state** and **distance-vector approaches**:
  - **Message complexity**:
    - **<u>LS:</u>** with N nodes & E links - O(N*E) messages sent.
    - **<u>DV:</u>** exchange between neighbors only.
  - **Speed of convergence**:
    - **<u>LS:</u>** O(N^2) algorithm requires O(N*E) messages.
      - May have oscillations.
    - **<u>DV:</u>** convergence time varies.
      - May have routing loops.
      - Count-to-infinity problem.
  - **Robustness**:
    - **<u>LS</u>**: node can advertise incorrect **link** cost.
      - Each node **computes** only its **own table**.
    - **<u>DV</u>**: node can advertise incorrect **path** cost.
      - Each node's table **used** by **others**.
        - **Error propagates** thru network.

HIERARCHICAL ROUTING

- **Routing** solely based on **LS** or **DV** algorithms is not **feasible** for current state of the **Internet**.
  - **Scale: billions of destinations.**
    - Cannot store all destinations in routing tables.
    - Routing tables exchange would swamp the links.
  - **Administrative autonomy.**
    - Internet = network of networks.
    - Each network might be controlling routing differently.
- <u>**Solution**</u> – **autonomous systems** (**AS**).
  - Routers **aggregated** into regions ("*domains*")

HIERARCHICAL ROUTING: AUTONOMOUS SYSTEMS (1)

- **Autonomous system** (**AS**) – **group** of **routers** that are under the same **admin control**.
  - **Intra-AS** routing protocol governs routing **inside AS**.
  - **Inter-AS** routing protocol governs routing **between AS’es**.
    - **Gateway routers** are used to **connect** multiple AS’es.



Interconnected autonomous systems

HIERARCHICAL ROUTING: AUTONOMOUS SYSTEMS (2)

- **Intra-AS routing:**
  - **Routing** among hosts and routers in **same AS** (“*network*”).
  - All **routers** in **AS** must run **same intra-domain** routing protocol.
  - **Routers** in **different AS** can run **different intra-domain** routing protocols.
  - **Gateway router** is at the “**edge**” of its own AS.
    - Has link(s) to router(s) in other AS’es.
- **Inter-AS routing:**
  - **Routing** among **different** AS’es.
  - **Communicating AS’es** must run **same inter-domain** routing algorithms.
- **Forwarding table** in **gateway routers** is configured by both: **intra-AS** & **inter-AS** routing.
  - **Intra-AS** routing determines entries for destinations **within AS**.
  - **Inter-AS** & **intra-AS** determine entries for **external** destinations.

HIERARCHICAL ROUTING: PROTOCOLS

- Most **common intra-AS** routing **protocols**:
  - Routing Information Protocol (**RIP**).
  - **Open Shortest Path First (OSPF).**
  - Interior Gateway Routing Protocol (**IGRP**).
    - Cisco proprietary.
- **Internet's inter-AS** routing **protocol**:
  - Border Gateway Protocol (**BGP**).

INTRA-AS ROUTING: OSPF (1)

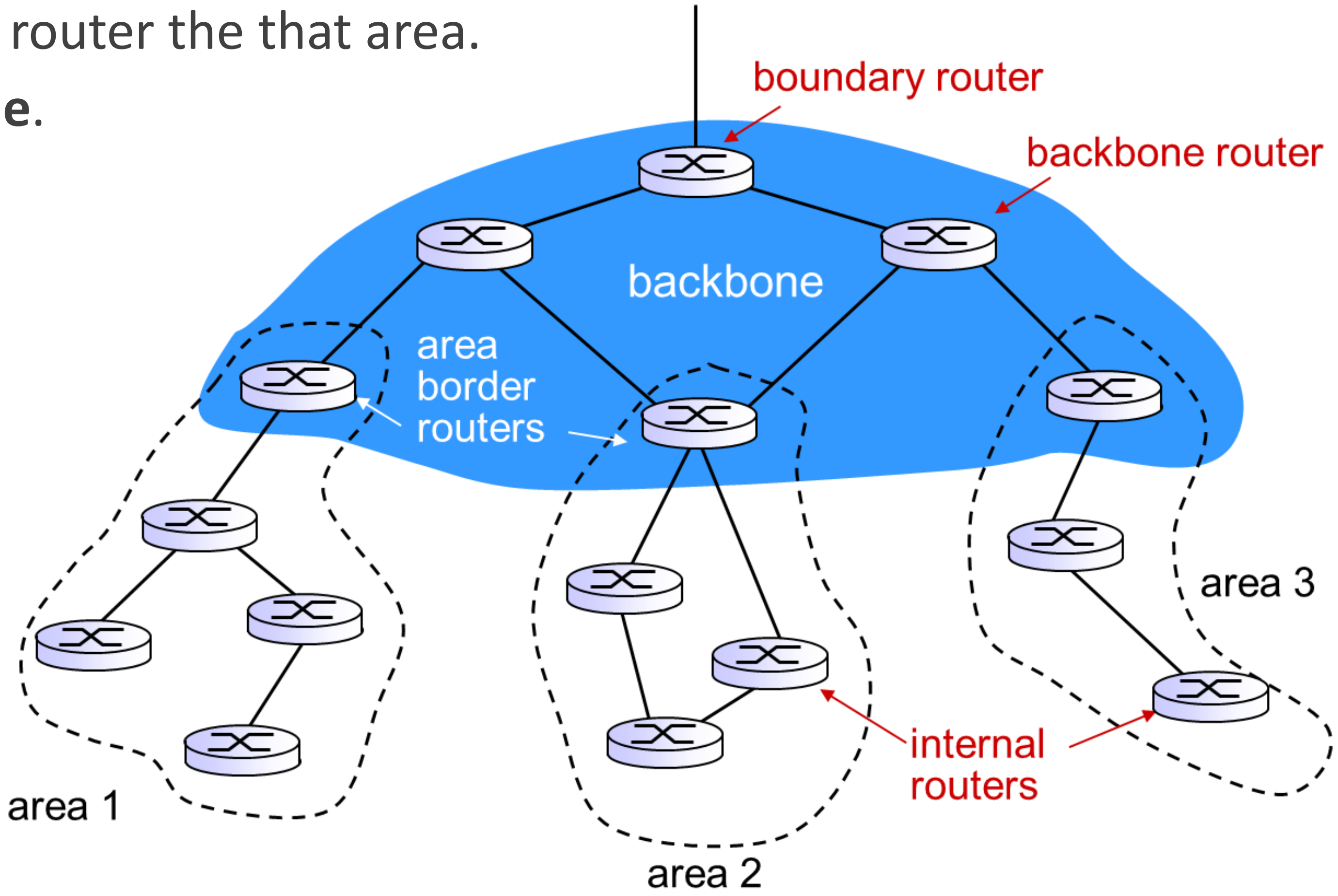
- Open Shortest Path First (**OSPF**) – **intra-AS link-state** protocol.
- **Main principles** of OSPF:
  - **Link-state information** is **distributed** using **flooding** approach (*discussed later*).
    - Carried in **OSPF messages** directly over **IP**.
    - Implements its own **reliable transfer**.
  - Each **router** constructs **complete network graph** of entire **AS**.
  - **Dijkstra algorithm** is used to find **least-cost paths**.

INTRA-AS ROUTING: OSPF (2)

- **OSPF** additional **advanced services**:
  - Provides **security**.
    - OSPF messages are authenticated.
  - Allows using **multiple** same-cost **paths**.
  - Links have **multiple cost** metrics for **different type-of-service**.
  - Supports hierarchical routing in large AS.

INTRA-AS ROUTING: OSPF (3)

- **OSPF** allows arranging **AS hierarchically** into **areas**.
  - Each **area** runs its **own OSPF algorithm**.
  - Each **router broadcasts** its **link state** to **all** router the that area.
  - Two-level hierarchy: **local area** & **backbone**.
- **Components** of an **AS area**:
  - **Area border routers.**
    - Responsible for routing packets outside the area.
  - **Backbone routers.**
    - Run OSPF routing limited to backbone.
  - **Boundary routers.**
    - Connect to other AS'es.

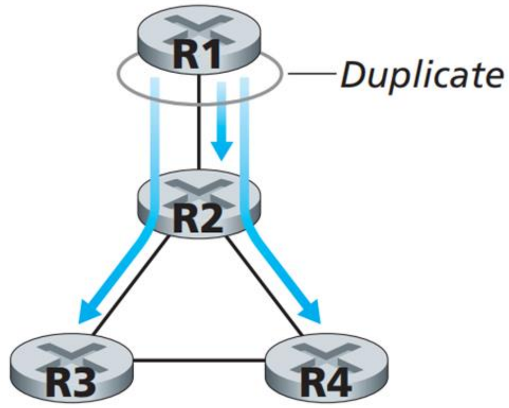


BROADCAST ROUTING (1)

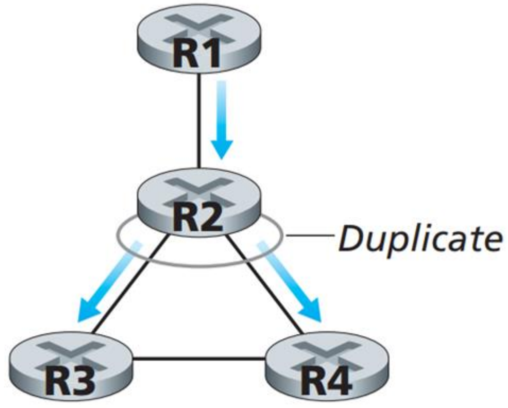
- **Network layer** provides three **packet delivery methods**:
  - **Unicasting.**
    - Single source to single destination delivery.
  - **Broadcasting.**
    - Single source to all other nodes in the network.
  - **Multicasting.**
    - Single source to a subset of nodes in the network.

BROADCAST ROUTING (2)

- **Broadcasting approaches:**
  - **N-way unicast.**
    - Source sends a copy of packet to each destination.
    - Suffers from inefficiency.
    - Requires complex process of obtaining addresses of all recipients.
  - **Uncontrolled flooding.**
    - Source sends a copy of packet to all of its neighbors.
    - Issues: cycles & broadcast storm.
  - **Controlled flooding.**
    - Nodes avoid broadcasting copies of packets.
  - **Spanning-tree broadcast.**
    - Assures no redundant packets received by any node.



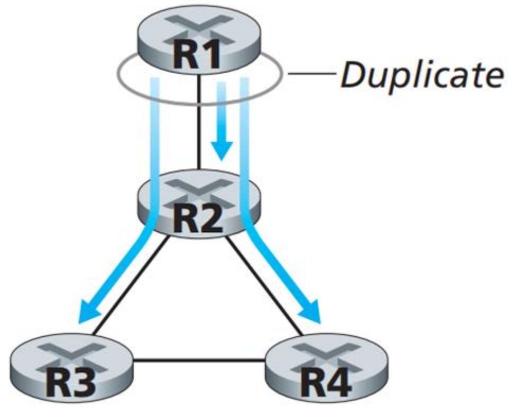
Source-duplication



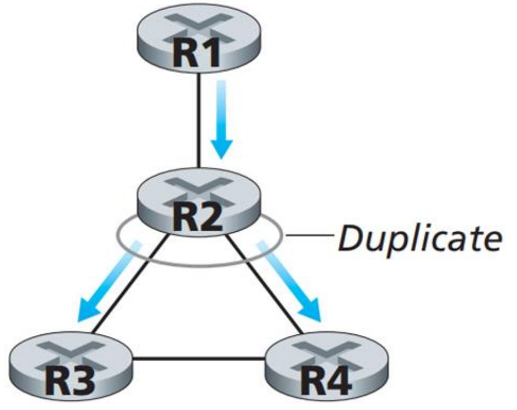
In-network duplication

BROADCAST ROUTING: UNCONTROLLED FLOODING

- **Principles** of **uncontrolled flooding**:
  - **Source** sends a **copy** of packet to **all neighbors**.
  - **Node** receives **broadcast packet** $\rightarrow$ **duplicates** & **forwards** to all of its **neighbors**
    - **Except** the neighbor from which packet **was received**.
- **Flaws** of **uncontrolled flooding**:
  - **Cycles** in network – copies of packets could circulate **indefinitely**.
  - **Broadcast storm** – multiple copies of packets eventually **clog** the **network**.



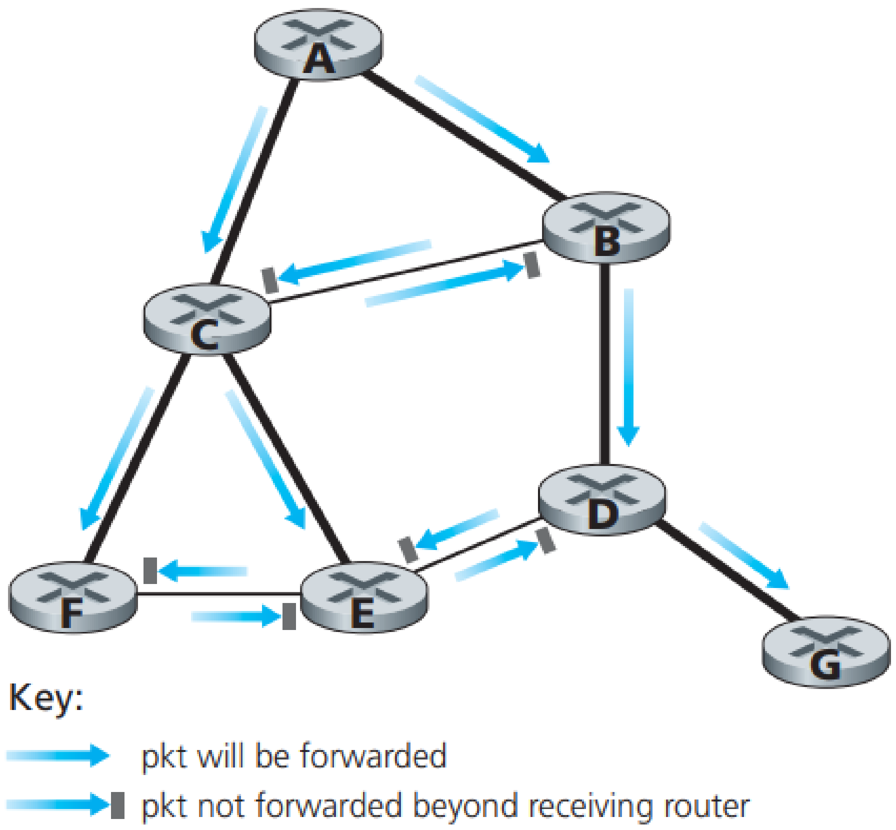
Source-duplication



In-network duplication

BROADCAST ROUTING: CONTROLLED FLOODING

- **Controlled flooding** is used to **mitigate** a **broadcast storm** issue of uncontrolled flooding.
- Two approaches to **controlled flooding**:
  - **Sequence-number-controlled flooding**.
    - Uses source node address (any unique id) & broadcast sequence number.
    - Each node maintains a list of already received (duplicated/forwarded) broadcast packets.
      - Newly arrived packets that are in the list – dropped.
  - **Reverse path forwarding (RPF)**.
    - Transmits packet copy on all of the links, only when packet arrived on the link that is on the shortest path.
    - Discards the packet otherwise.

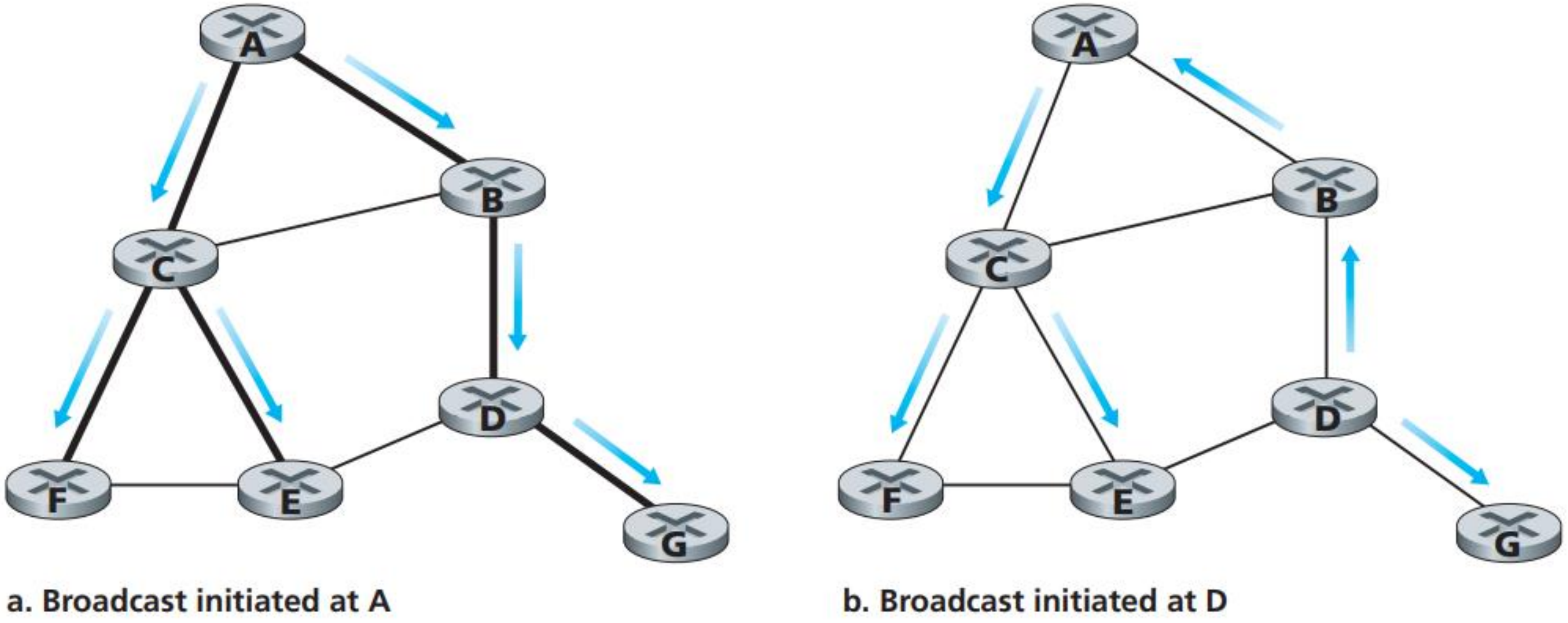


Reverse path forwarding

BROADCAST ROUTING: SPANNING TREE

- **Spanning tree** broadcast is used to **mitigate** transmission of **redundant** broadcast packets.
  - **Broadcast packets** are only **sent** along the nodes in **network's spanning tree**.
    - **Spanning tree** contains **each** and **every** node in the **graph** with no **cycles**.
  - Network **spanning tree** allows to **begin** flooding from **any node** in the network.

- Several **greedy algorithms** were proposed for network spanning tree **construction**:
  - **Boruvka's** algorithm**.**
  - **Prim's** algorithm**.**
  - **Kruskal's** algorithm**.**



Broadcast along a spanning tree

SUMMARY

- Goal of routing protocols.
- Network as a graph.
- Routing algorithms classification.
- Link-state routing.
- Dijkstra’s algorithm.
- Bellman-Ford equation.
- Distance-vector algorithm.
- Comparison of LS & DV algorithms.
- Intra-AS & inter-AS routing.
- Open Shortest Path First (OSPF).
- Broadcast & flooding.