CS 268: Lecture 9
Intra-domain Routing Protocols

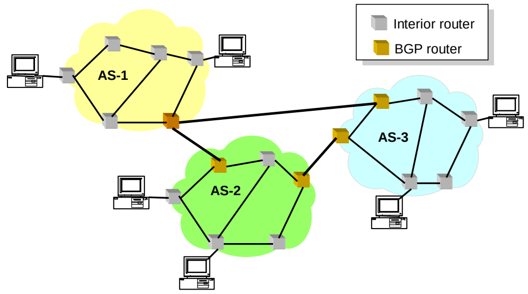
Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

(*Based in part on Aman Shaikh's slides)

Internet Routing

- Internet organized as a two level hierarchy
- First level – autonomous systems (AS's)
  - AS – region of network under a single administrative domain
- AS's run an intra-domain routing protocols
  - Distance Vector, e.g., Routing Information Protocol (RIP)
  - Link State, e.g., Open Shortest Path First (OSPF)
- Between AS's runs inter-domain routing protocols, e.g., Border Gateway Routing (BGP)
  - De facto standard today, BGP-4

Example



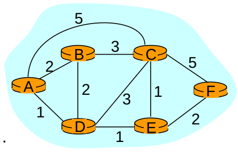
Intra-domain Routing Protocols

- Based on unreliable datagram delivery
- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - Each neighbor periodically exchange reachability information to its neighbors
- Link state
  - Open Shortest Path First (OSPF), based on Dijkstra
  - Each network periodically floods immediate reachability information to other routers

---

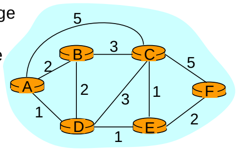
Routing

- Goal: determine a "good" path through the network from source to destination
  - Good means usually the shortest path
- Network modeled as a graph
  - Routers → nodes
  - Link →edges
    - Edge cost: delay, congestion level,…



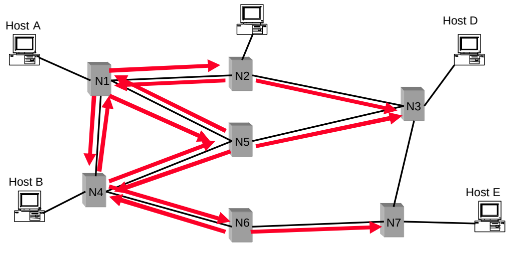
Routing Problem

- Assume
  - A network with N nodes, where each edge is associated a cost
  - A node knows only its neighbors and the cost to reach them
- How does each node learns how to reach every other node along the shortest path?

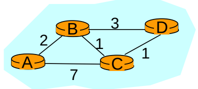


Distance Vector: Control Traffic

- When the routing table of a node changes, the node sends its table to its neighbors
- A node updates its table with information received from its neighbors



Example: Distance Vector Algorithm



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

```
Initialization:
  for all neighbors V do
    if V adjacent to A
      D(A, V) = c(A,V);
  else
      D(A, V) = ∞;
…
```

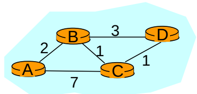
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

(D(C,A), D(C,B), D(C,D))

```
…
loop:
…
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```

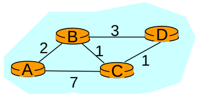
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	C

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A, D) = min(D(A, D), D(A, C) + D(C,D))
= min(∞ , 7 + 1) = 8

(D(C,A), D(C,B), D(C,D))

```
…
loop:
…
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```

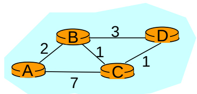
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	C

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

```
…
loop:
…
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```

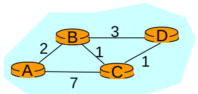
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (B→A, C→A)



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A,D) = min(D(A,D), D(A,B) + D(B,D))
= min(8, 2 + 3) = 5

D(A,C) = min(D(A,C), D(A,B) + D(B,C))
= min(7, 2 + 1) = 3

```
…
loop:
…
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```

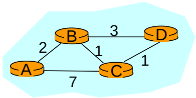
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

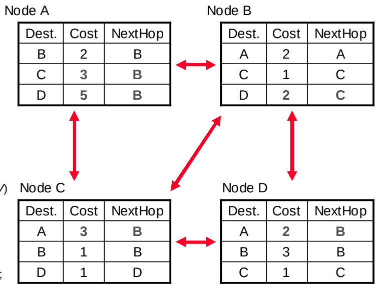
Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: End of 1st Iteration



```
...
loop:
...
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```



Node A

Dest.	Cost	NextHop
B	2	B
C	**3**	**B**
D	**5**	**B**

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	**2**	**C**

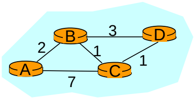
Node C

Dest.	Cost	NextHop
A	**3**	**B**
B	1	B
D	1	D

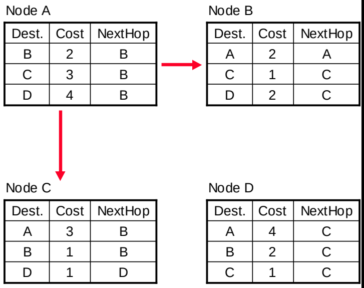
Node D

Dest.	Cost	NextHop
A	**2**	**B**
B	3	B
C	1	C

Example: End of 3nd Iteration



```
...
loop:
...
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

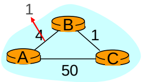
Node D

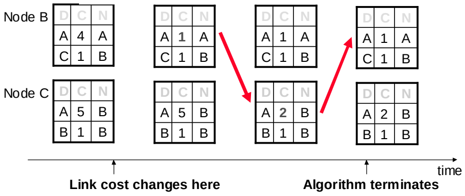
Dest.	Cost	NextHop
A	4	C
B	2	C
C	1	C

Nothing changes → algorithm terminates

Distance Vector: Link Cost Changes

```
loop:
  wait (link cost update or update message)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y) =  D(A,Y) + d
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors
forever
```





	D	C	N	D	C	N	D	C	N	D	C	N
Node B	A	4	A	A	**1**	A	A	1	A	A	1	A
	C	1	B	C	1	B	C	1	B	C	1	B
Node C	A	5	B	A	5	B	A	**2**	B	A	2	B
	B	1	B	B	1	B	B	1	B	B	1	B

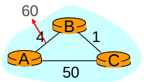
Link cost changes here

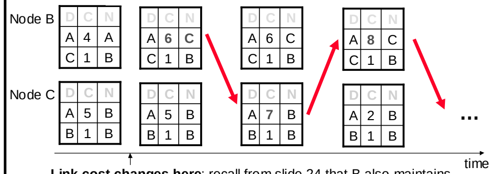
Algorithm terminates

"good news travels fast"

Distance Vector: Count to Infinity Problem

```
loop:
  wait (link cost update or update message)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y) =  D(A,Y) + d
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors
forever
```





	D	C	N	D	C	N	D	C	N	D	C	N
Node B	A	4	A	A	**6**	**C**	A	6	C	A	**8**	C
	C	1	B	C	1	B	C	1	B	C	1	B
Node C	A	5	B	A	5	B	A	**7**	B	A	2	B
	B	1	B	B	1	B	B	1	B	B	1	B

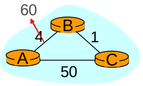
…

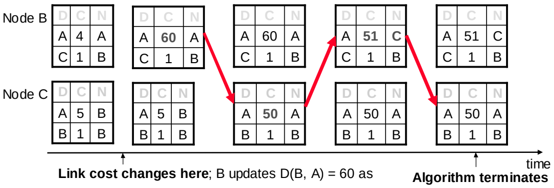
"bad news travels slowly"

Link cost changes here; recall from slide 24 that B also maintains shortest distance to A through C, which is 6. Thus D(B, A) becomes 6 !

Distance Vector: Poisoned Reverse

- If C routes through B to get to A:
  - C tells B its (C's) distance to A is infinite (so B won't route to A via C)
  - Will this completely solve count to infinity problem?





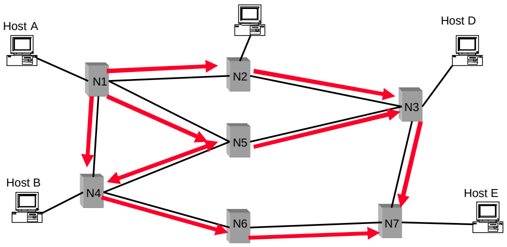
	D	C	N	D	C	N	D	C	N	D	C	N	D	C	N
Node B	A	4	A	A	**60**	A	A	60	A	A	**51**	**C**	A	51	C
	C	1	B	C	1	B	C	1	B	C	1	B	C	1	B
Node C	A	5	B	A	5	B	A	**50**	A	A	50	A	A	50	A
	B	1	B	B	1	B	B	1	B	B	1	B	B	1	B

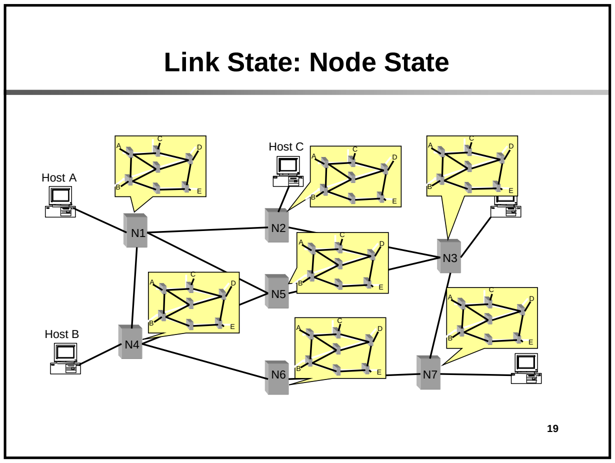
Link cost changes here; B updates D(B, A) = 60 as C has advertised D(C, A) = ∞

Algorithm terminates

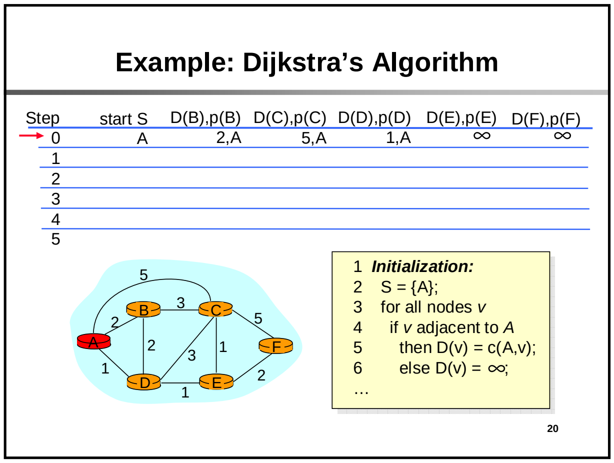
Link State: Control Traffic

- Each node floods its local information to every other node in the network
- Each node ends up knowing the entire network topology → use Dijkstra to compute the shortest path to every other node





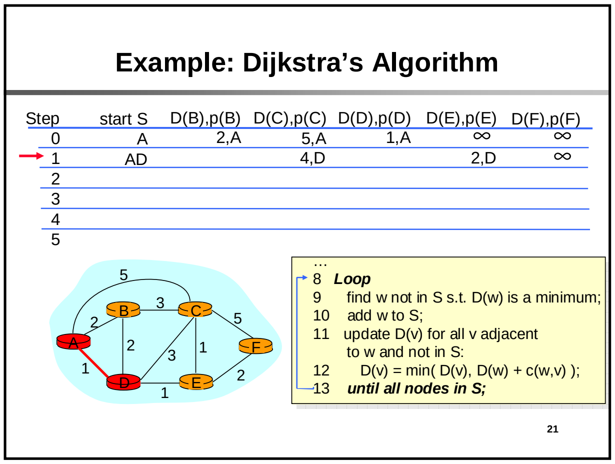
Link State: Node State



Example: Dijkstra’s Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→ 0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						

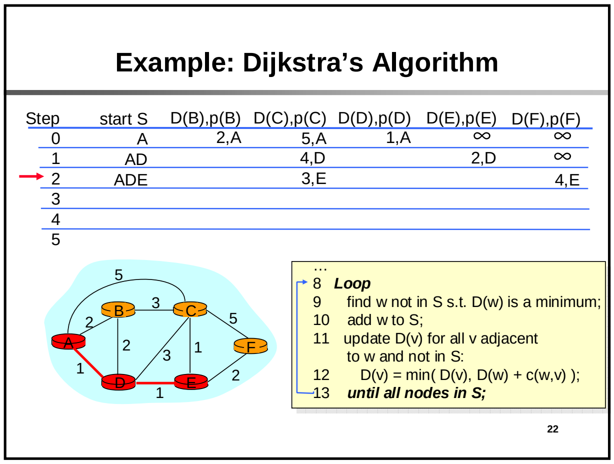
```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;
…
```



Example: Dijkstra’s Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
→ 1	AD		4,D		2,D	∞
2						
3						
4						
5						

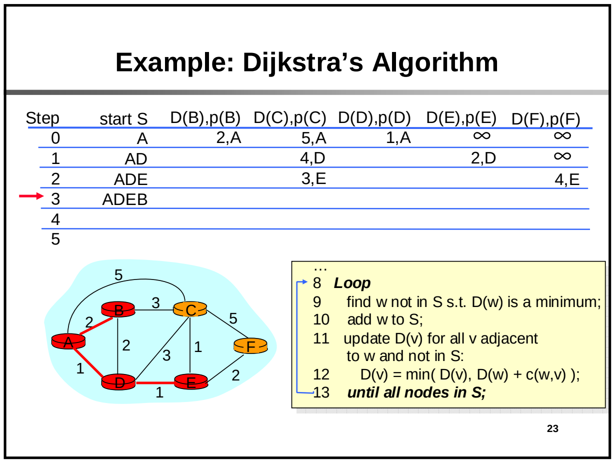
```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
until all nodes in S;
```



Example: Dijkstra’s Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
→ 2	ADE		3,E			4,E
3						
4						
5						

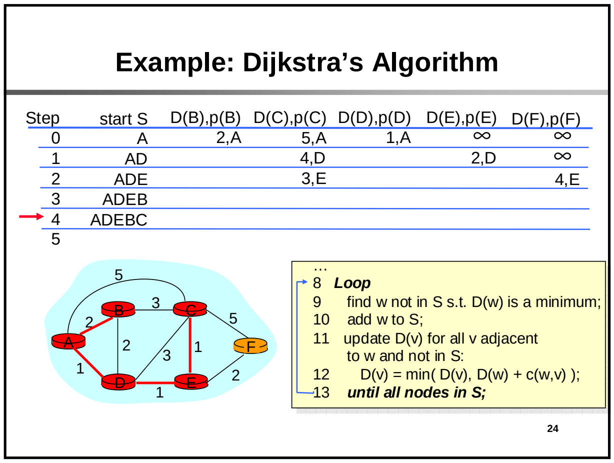
```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
until all nodes in S;
```



Example: Dijkstra’s Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
→ 3	ADEB					
4						
5						

```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
until all nodes in S;
```



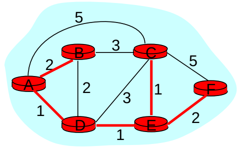
Example: Dijkstra’s Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
→ 4	ADEBC					
5						

```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
until all nodes in S;
```

Example: Dijkstra's Algorithm

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
→ 5	ADEBCF					



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
    to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
until all nodes in S;
```

Link State vs. Distance Vector

Message complexity
- LS: O(n^2*e) messages
  - n: number of nodes
  - e: number of edges
- DV: O(d*n*k) messages
  - d: node's degree
  - k: number of rounds

Time complexity
- LS: O(n*log n)
- DV: O(n)

Convergence time
- LS: O(1)
- DV: O(k)

Robustness: what happens if router malfunctions?
- LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table
- DV:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagate through network

Open Shortest Path First (OSPF)

- All routers in the domain come to a consistent view of the topology by exchange of Link State Advertisements (LSAs)
- Router describes its local connectivity (i.e., set of links) in an LSA
  - Set of LSAs (self-originated + received) at a router = topology
- Hierarchical routing
  - OSPF domain can be divided into areas
  - Hub-and-spoke topology with area 0 as hub and other non-zero areas as spokes

OSPF Performance

- OSPF processing impacts convergence, (in)stability
  - Load is increasing as networks grow
- Bulk of OSPF processing is due to LSAs
  - Sending/receiving LSAs
  - LSAs can trigger Route calculation (Dijkstra's algorithm)
- Understanding dynamics of LSA traffic is key for a better understanding of OSPF

Objectives for OSPF Monitor

- Real-time analysis of OSPF behavior
  - Trouble-shooting, alerting, validation of maintenance
  - Real-time snapshots of OSPF network topology
- Off-line analysis
  - Post-mortem analysis of recurring problems
  - Generate statistics and reports about network performance
  - Identify anomaly signatures
  - Facilitate tuning of configurable parameters
  - Analyze OSPF behavior in commercial networks

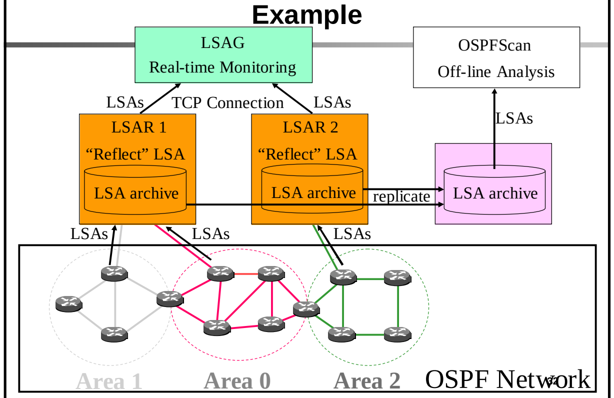
Categorizing LSA Traffic

- A router originates an LSA due to…
  - Change in network topology — **Change LSAs**
    - Example: link goes down or comes up
    - Detection of anomalies and problems
  - Periodic soft-state refresh — **Refresh LSAs**
    - Recommended value of interval is 30 minutes
    - Forms baseline LSA traffic
- LSAs are disseminated using reliable flooding
  - Includes change and refresh LSAs
  - Flooding leads to duplicate copies of LSAs being received at a router — **Duplicate LSAs**
  - Overhead: wastes resources

Components

- Data collection: LSA Reflector (LSAR)
  - Passively collects OSPF LSAs from network
  - “Reflects” streams of LSAs to LSAG
  - Archives LSAs for analysis by OSPFScan
- Real-time analysis: LSA aGgregator (LSAG)
  - Monitors network for topology changes, LSA storms, node flaps and anomalies
- Off-line analysis: OSPFScan
  - Supports queries on LSA archives
  - Allows playback and modeling of topology changes
  - Allows emulation of OSPF routing

Example



How LSAR attaches to Network

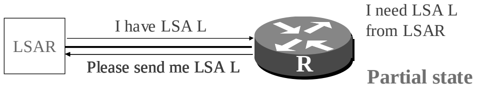
- Host mode: Join multicast group
- Full adjacency mode: form full adjacency (= peering session) with a router
- Partial adjacency mode: keep adjacency in a state that allows LSAR to receive LSAs, but does not allow data forwarding over link

---

How LSAR attaches to Network

- Host mode
  - Join multicast group
  - **Adv:** completely passive
  - **Disadv:** not reliable, delayed initialization of LSDB
- Full adjacency mode
  - Form full adjacency (= peering session) with a router
  - **Adv:** reliable, immediate initialization of LSDB
  - **Disadv:** LSAR’s instability can impact entire network
- Partial adjacency mode
  - Keep adjacency in a state that allows LSAR to receive LSAs, but does not allow data forwarding over link
  - **Adv:** reliable, LSAR’s instability does not impact entire network, immediate initialization of LSDB
  - **Disadv:** can raise alarms on the router

Partial Adjacency for LSAR

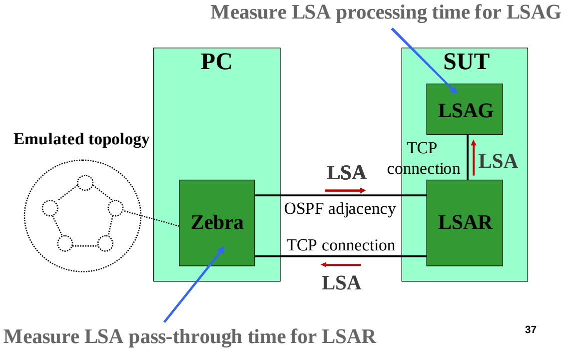


- Router R does not advertise a link to LSAR
- LSAR does not originate any LSAs
- Routers (except R) not aware of LSAR’s presence
  - Does not trigger routing calculations in network
  - LSAR’s going up/down does not impact network
- LSAR↔R link is not used for data forwarding

Performance Evaluation

- Performance of LSAR and LSAG through lab experiments
  - LSAR and LSAG are key to real-time monitoring
- How performance scales with LSA-rate and network size

Experimental Setup



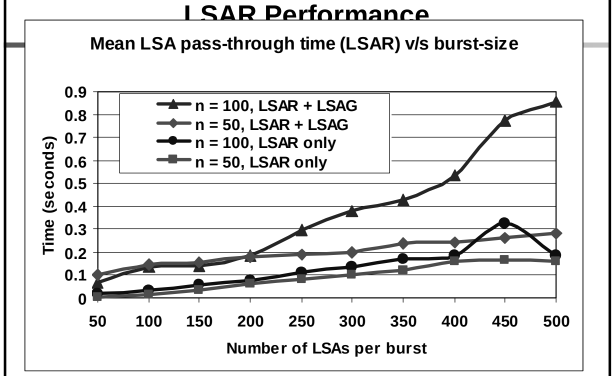
Measure LSA processing time for LSAG

Measure LSA pass-through time for LSAR

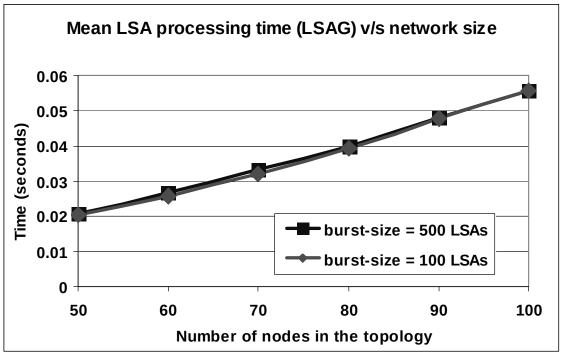
Methodology

- Send a burst of LSAs from Zebra to LSAR
  - Vary number of LSAs (l) in a burst of 1 sec duration
- Use of fully connected graph as the emulated topology
  - Vary number of nodes (n) in the topology
- Performance measurements
  - LSAR performance: LSA “pass-through” time
    - Zebra measures time difference between sending and receiving an LSA from LSAR
  - LSAG performance: LSA processing time
    - Instrumentation of LSAG code

LSAR Performance



LSAG Performance

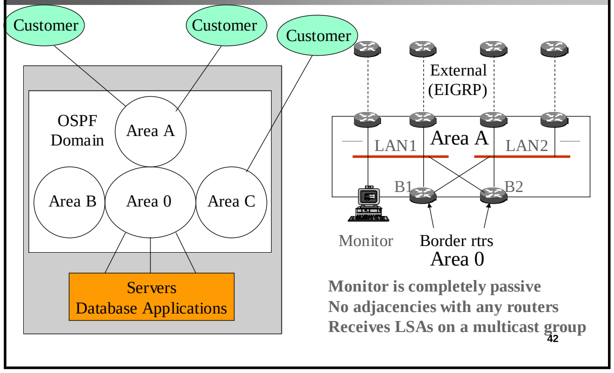


Enterprise Network Case Study

- The network provides customers with connectivity to applications and databases residing in the data center
- OSPF network
  - 15 areas, 500 routers
    - This case study covers 8 areas, 250 routers
    - One month: April 2002
  - Link-layer = Ethernet-based LANs
- Customers are connected via leased lines
  - Customer routes are injected via EIGRP into OSPF
    - The routes are propagated via external LSAs
    - Quite reasonable for the enterprise network in question

---

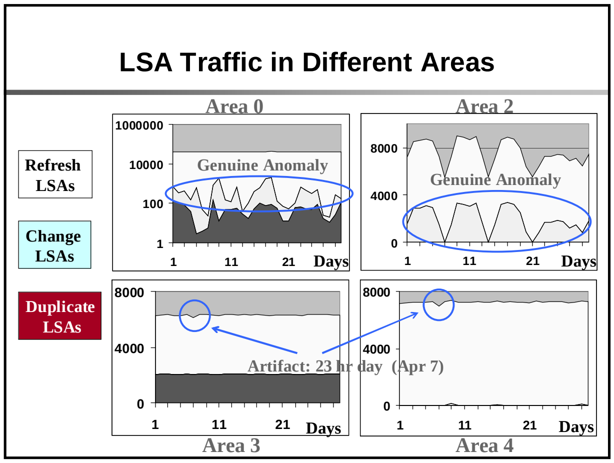
Enterprise Network Topology



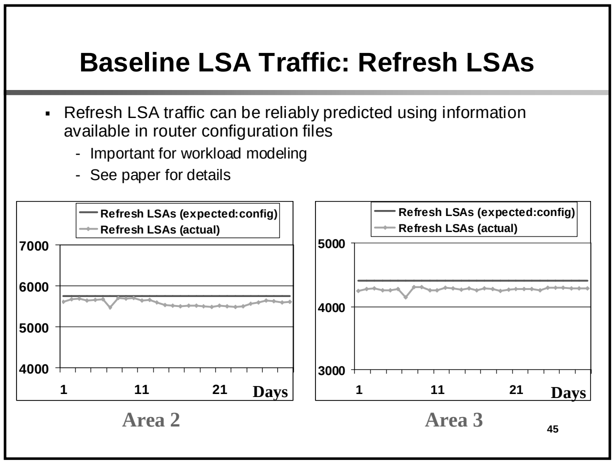
Monitor is completely passive
No adjacencies with any routers
Receives LSAs on a multicast group

Highlights of the Results

- Categorize, baseline and predict
  - Categories: Refresh, Change, Duplicate; External, Internal
  - Bulk of LSA traffic is due to refresh
  - Refresh LSA traffic is smooth: no evidence of refresh synchronization across network
  - Refresh LSA traffic is predictable from router configuration info
- Detect, diagnose and act
  - Almost all LSAs arise from persistent yet partial failure modes
  - Internal LSA spikes
    - Indicate router hardware degradation
    - Carry out preventive maintenance
  - External LSA spikes
    - Indicate degradation in customer connectivity
    - Call customer before customer calls you
- Propose Improvements
  - Simple configuration changes to reduce duplicate LSA traffic

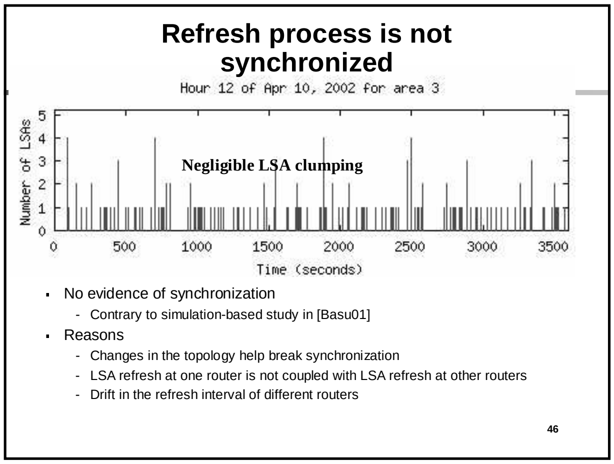


LSA Traffic in Different Areas



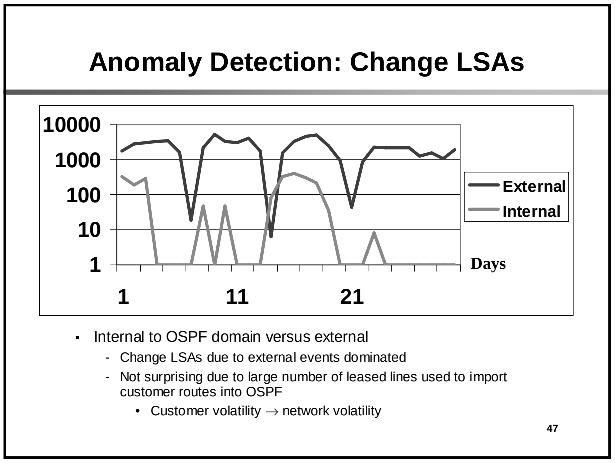
Baseline LSA Traffic: Refresh LSAs

- Refresh LSA traffic can be reliably predicted using information available in router configuration files
  - Important for workload modeling
  - See paper for details



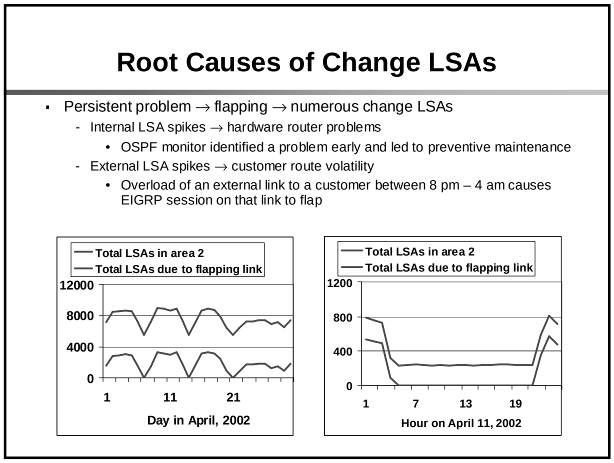
Refresh process is not synchronized

- No evidence of synchronization
  - Contrary to simulation-based study in [Basu01]
- Reasons
  - Changes in the topology help break synchronization
  - LSA refresh at one router is not coupled with LSA refresh at other routers
  - Drift in the refresh interval of different routers



Anomaly Detection: Change LSAs

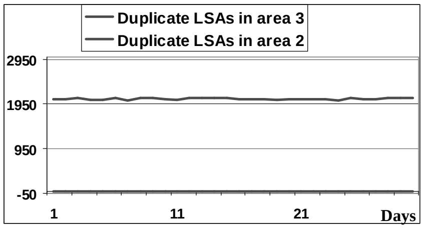
- Internal to OSPF domain versus external
  - Change LSAs due to external events dominated
  - Not surprising due to large number of leased lines used to import customer routes into OSPF
    - Customer volatility → network volatility



Root Causes of Change LSAs

- Persistent problem → flapping → numerous change LSAs
  - Internal LSA spikes → hardware router problems
    - OSPF monitor identified a problem early and led to preventive maintenance
  - External LSA spikes → customer route volatility
    - Overload of an external link to a customer between 8 pm – 4 am causes EIGRP session on that link to flap

Overhead: Duplicate LSAs

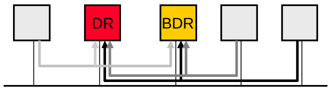


- Why do some areas witness substantial duplicate LSA traffic, while other areas do not witness any?
  - OSPF flooding over LANs leads to control plane asymmetries and to imbalances in duplicate LSA traffic

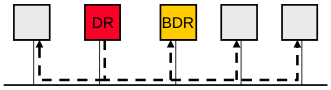
---

OSPF Operations over Broadcast Networks

1) Each node sends an LSA to multicast group *DR-rtrs*
   - Both designated router (DR) and backup designated router BDR subscribe to this group



2) DR floods the LSA back to all routers on the network
   - Send to *all-rtrs* multicast group to which all nodes subscribe

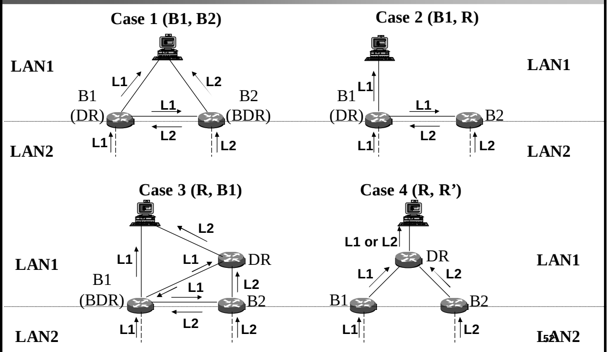


Control Plane Asymmetry

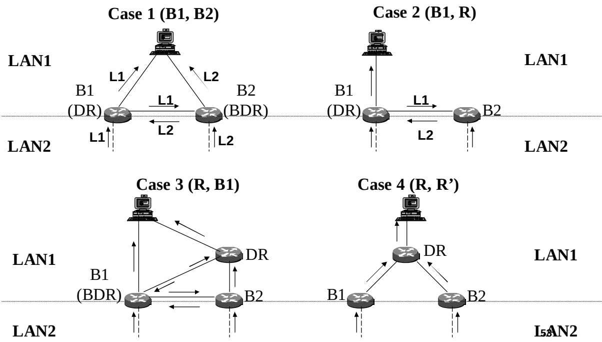
- Two LANs (LAN1 and LAN2) in each area
- Monitor is on LAN1
- Routers B1 and B2 are connected to LAN1 and LAN2
- LSAs originated on LAN2 can get duplicated depending on which routers have become DR and BDR on LAN1
  - Leads to control plane asymmetry
  - Four cases

- Note: if a BDR receives an LSA on another interface, it floods the LSA to all nodes (i.e., it sends the LSA to the *all-rtrs* address)

Four Cases



Four Cases



Eliminating Duplicate LSA Traffic

	Case1	Case 2	Case 3	Case 4
Duplicate LSA traffic	High	None	High	None
Deterministic via configuration	Yes	No	No	Yes
Area 2		X		X configuration change
Area 3			X	X configuration change

Summary

- Categorize and baseline LSA traffic
  - Refresh LSAs: constitute bulk of overall LSA traffic
    - No evidence of synchronization between different routers
    - Refresh LSA traffic predictable from configuration information
- Detect, diagnose and act on anomalies
  - Change LSAs: can indicate persistent yet partial failure modes
    - Internal LSA spikes → hardware router problems → preventive router maintenance
    - External LSA spikes → customer congestion problems → “preventive” customer care
- Propose changes to improve performance
  - Duplicate LSAs: can arise from control plane asymmetries
    - Simple configuration changes can eliminate duplicate LSAs and improve performance

Other Problems Caught

- Configuration problem
  - Identified assignment of same router-id to two routers in enterprise network
- OSPF implementation bug
  - Caught a bug in type-3 LSA generation code of a router vendor in ISP network
    - Faster refresh of LSAs than standards-mandated rate

LSA aGregator (LSAG)

- Analyzes “reflected” LSAs from LSARs in real-time
- Generates console messages:
  - Change in OSPF network topology
    - ADJACENY COST CHANGE: rtr 10.0.0.1 (intf 10.0.0.2) → rtr 10.0.0.5 old_cost 1000 new_cost 50000 area 0.0.0.0
  - Node flaps
    - RTR FLAP: rtr 10.0.0.12 no_flaps 7 flap_window 570 sec
  - LSA storms
    - LSA STORM: lstype 3 lsid 10.1.0.0 advrt 10.0.0.3 area 0.0.0.0 no_lsas 7 storm_window 470 sec
  - Anomalous behavior
    - TYPE-3 ROUTE FROM NON-BORDER RTR: ntw 10.3.0.0/24 rtr 10.0.0.6 area 0.0.0.0
- Dumps snapshots of network topology

OSPFScan

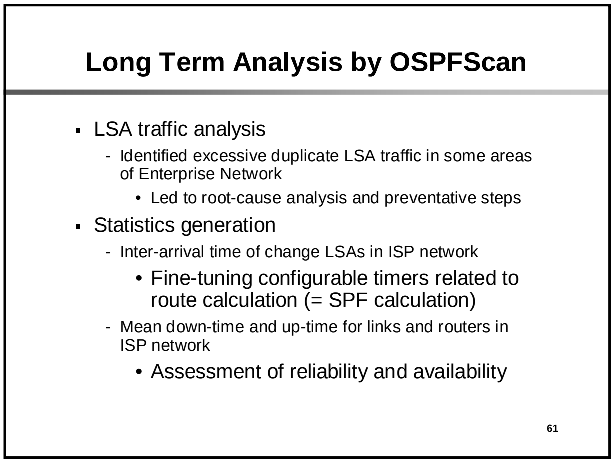
- Tools for off-line analysis of LSA archives
  - Parse, select (based on queries), and analyze
- Functionality supported by OSPFScan
  - Classification of LSA traffic
    - Change LSAs, refresh LSAs, duplicate LSAs
  - Emulation of OSPF Routing
    - How OSPF routing tables evolved in response to network changes
    - How end-to-end path within OSPF domain looked like at any instance
  - Modeling of topology changes
    - Vertex addition/deletion and link addition/deletion/change_cost
  - Playback of topology change events
  - Statistics and report generation

Deployment

- Tier-1 ISP network
  - Area 0, 100+ routers; point-to-point links
  - Deployed since January, 2003
  - LSA archive size: 8 MB/day
  - LSAR connection: partial adjacency mode
- Enterprise network
  - 15 areas, 500+ routers; Ethernet-based LANs
  - Deployed since February, 2002
  - LSA archive size: 10 MB/day
  - LSAR connection: host mode

LSAG in Day-to-day Operations

- Generation of alarms by feeding messages into higher layer network management systems
  - Grouping of messages to reduce the number of alarms
  - Prioritization of messages
- Validation of maintenance steps and monitoring the impact of these steps on network-wide OSPF behavior
  - Example:
    - Network operators use cost-out/cost-in of links to carry out maintenance
    - A “link-audit” web-page allows operators to keep track of link costs in real-time



Long Term Analysis by OSPFScan

- LSA traffic analysis
  - Identified excessive duplicate LSA traffic in some areas of Enterprise Network
    - Led to root-cause analysis and preventative steps
- Statistics generation
  - Inter-arrival time of change LSAs in ISP network
    - Fine-tuning configurable timers related to route calculation (= SPF calculation)
  - Mean down-time and up-time for links and routers in ISP network
    - Assessment of reliability and availability