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 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

Example: Distance Vector Algorithm

Example: 1st Iteration (C → A)

Example: 1st Iteration (C → A)

Example: 1st Iteration (B → A, C → A)

Example: 1st Iteration (B → A, C → A)
Example: End of 1st Iteration

Node A

Node B

7 Loop:
12 if (update D(v, y) received from y)
13 for all destinations y do
14 if (destination y through v)
15 D(A, y) = min(D(A, y) + D(v, y))
16 else
17 D(A, y) = D(A, y)
18 if (there is a new minimum for dest. y)
19 send(D(A, y)) to all neighbors
20 forever

Distance Vector: Link Cost Changes

7 Loop:
1 if (link update or update message)
2 for all destinations y do
3 if (D(A, y) changes by d)
4 D(A, y) = D(A, y) + d
5 else
6 D(A, y) = D(A, y)
7 if (there is a new minimum for destination y)
8 send(D(A, y)) to all neighbors
9 forever

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?

Distance Vector: Count to Infinity Problem

7 Loop:
1 if (link update or update message)
2 for all destinations y do
3 if (D(A, y) changes by d)
4 D(A, y) = D(A, y) + d
5 else
6 D(A, y) = D(A, y)
7 if (there is a new minimum for destination y)
8 send(D(A, y)) to all neighbors
9 forever

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

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>4,D</td>
<td>2,D</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loop
9. find w not in S s.t. D(w) is a minimum;
10. add w to S;
11. update D(v) for all v adjacent to w and not in S;
12. D(v) = min( D(v), D(w) + c(w,v) );
13. until all nodes in S;

Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>4,D</td>
<td>2,D</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loop
9. find w not in S s.t. D(w) is a minimum;
10. add w to S;
11. update D(v) for all v adjacent to w and not in S;
12. D(v) = min( D(v), D(w) + c(w,v) );
13. until all nodes in S;

Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>4,D</td>
<td>2,D</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loop
9. find w not in S s.t. D(w) is a minimum;
10. add w to S;
11. update D(v) for all v adjacent to w and not in S;
12. D(v) = min( D(v), D(w) + c(w,v) );
13. until all nodes in S;
Example: Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>start S</th>
<th>D(A),p(A)</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S</td>
<td>2, A</td>
<td>∞</td>
<td>5, A</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>4, D</td>
<td>∞</td>
<td>2, D</td>
<td>∞</td>
<td></td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>3, E</td>
<td>∞</td>
<td></td>
<td>4, E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loop

- 8. Loop
- 9. find w not in S s.t. D(w) is a minimum;
- 10. add w to S;
- 11. update D(v) for all v adjacent to w and not in S;
- 12. D(v) = min( D(v), D(w) + c(w,v) );
- 13. 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^2 * 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
    - Example: link goes down or comes up
  - Detection of anomalies and problems
  - Periodic soft-state refresh
    - 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
    - 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

How LSAR attaches to Network

- **Host mode:** Join multicast group

- **Full adjacency mode:** form full adjacency (= peer 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

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 pass-through time for LSAR

Methodology

- Send a burst of LSAs from Zebra to LSAR
  - Vary number of LSAs (n) 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

Mean LSA pass-through time (LSAR) v/s burst-size

LSAG Performance

Mean LSA processing time (LSAG) v/s network size

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

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

LSA Traffic in Different Areas

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

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

- 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

- Almost all LSAs arise from persistent yet partial failure modes

- Call customer before customer calls you

- Simple configuration changes to reduce duplicate LSA traffic
Overhead: Duplicate LSAs

![Graph showing Duplicate LSAs in area 3 over days 1 to 21.]

- 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

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-rts address)

OSPF Operations over Broadcast Networks

1) Each node sends an LSA to multicast group DR-rtts
   - 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-rtts multicast group to which all nodes subscribe

Four Cases

- Case 1 (B1, B2)
- Case 2 (B1, R)
- Case 3 (R, B1)
- Case 4 (R, R')

Eliminating Duplicate LSA Traffic

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate LSA traffic</td>
<td>High</td>
<td>None</td>
<td>High</td>
</tr>
<tr>
<td>Deterministic via configuration</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Area 2</td>
<td>X</td>
<td>X</td>
<td>configuration change</td>
</tr>
<tr>
<td>Area 3</td>
<td>X</td>
<td>X</td>
<td>configuration change</td>
</tr>
</tbody>
</table>
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 aGregaror (LSAG)

- Analyzes "reflected" LSAs from LSARs in real-time
- Generates console messages:
  - Change in OSPF network topology
    - ADJACENCY COST CHANGE: rtr 10.0.0.1 (old 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: lsid 10.1.0.0 advr 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: rtr 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