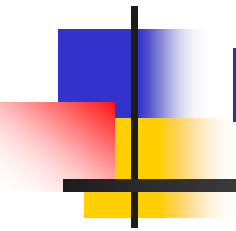
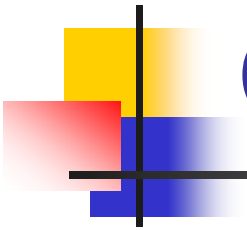
# A Distributed Adaptive Cache Update Algorithm for the Dynamic Source Routing Protocol



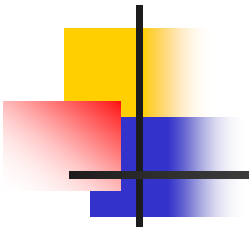
INFOCOM 2005

Presented by Tsung-Yuan Hsu



# Outline

- Introduction
- Dynamic Source Routing Protocol
- Problem Statement
- Definition of Cache Table
- Distributed Cache Update Algorithm
- Performance Evaluation
- Conclusion

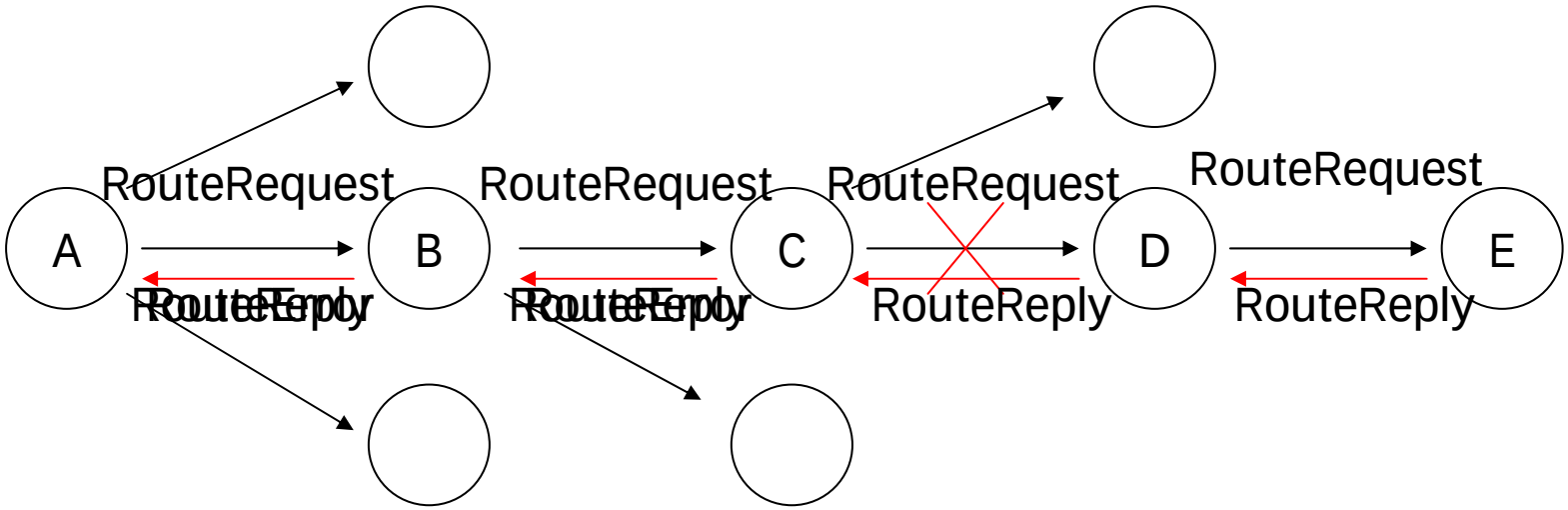


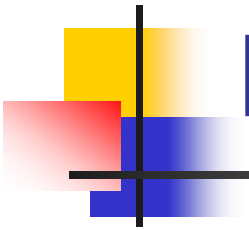
# Introduction

- Due to mobility, cached routes easily become stale.
- Using stale routes causes packet losses and increases latency and overhead.
- Authors investigate how to make on-demand routing protocols adapt quickly to topology changes.

# Dynamic Source Routing Protocol

- DSR consists of two on-demand mechanisms : Route Discovery and Route Maintenance.
- Route Discovery
- Route Maintenance





# Problem Statement

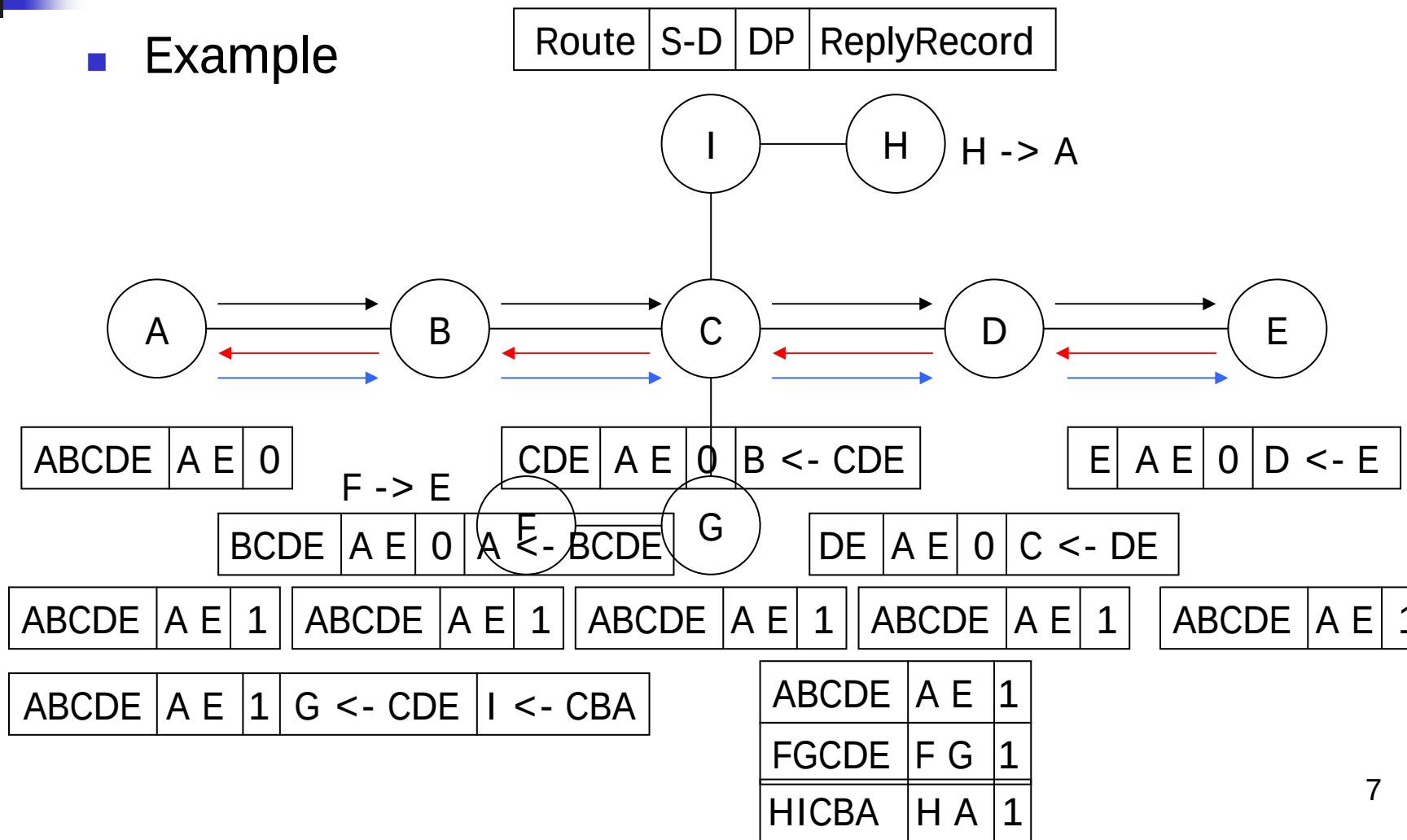
- On-demand Route Maintenance results in delayed awareness of mobility: a node is not notified when a cached route breaks until it uses the route to send packets.

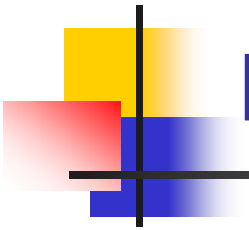
# Definition of Cache Table

- A cache table has no capacity limit.
  - Route: It stores the links starting from the current node to a destination or from a source to a destination.
  - SourceDestination: It is the source and destination pair.
  - DataPackets: It records whether the current node has forwarded 0, 1, or 2 data packets.
  - ReplyRecord: This field may contain multiple entries. Each entry records the neighbor to which a Route Reply is forwarded and the route starting from the current node to a destination.

# Definition of Cache Table

- Example





# Distributed Cache Update Algorithm

- We define a broken link as a forward or backward link.
  - Forward link: The flow using the route crosses the link in the same direction as the flow detecting the link failure.
  - Backward link: Otherwise.

# Distributed Cache Update Algorithm

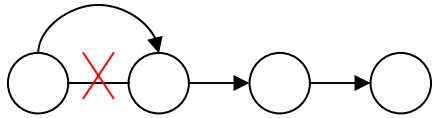
- If a route contains a forward link
  - If DataPackets is 0, then no downstream node needs to be notified, because the downstream nodes did not cache the link when forwarding a Route Reply.
  - If DataPackets is 1 or 2, then the upstream nodes need to be notified, because at least one packet has traversed the route and therefore the upstream nodes have cached the broken link.
  - If DataPackets is 2, then the downstream nodes need to be notified, because at least two data packets have traversed the route and thus the downstream nodes have cached the link.
  - If DataPackets is 1 and the route is different from the source route in the packet, then the downstream nodes need to be notified, because the downstream nodes cached the link when the first data packet traversed the route.
  - If DataPackets is 1 and the route is the same as the source route in the packet, then no downstream node needs to be notified, because the first data packet cannot be delivered and therefore the downstream nodes did not cache the link through forwarding packets with this route.

# Distributed Cache Update Algorithm

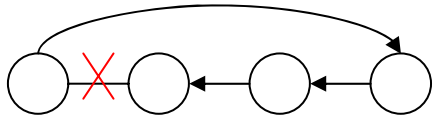
- If a route contains a backward link
  - Both downstream and upstream nodes need to be notified, because the route has been cached in the nodes.

# Distributed Cache Update Algorithm

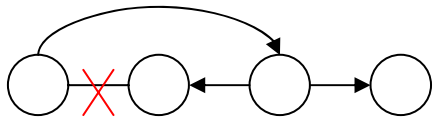
- If the link is a forward link
  - The node is upstream to it, then the algorithm notifies upstream neighbor.
  - The node is downstream to it, then
    1. If the node is the other endpoint of the link, then notify its downstream neighbor.

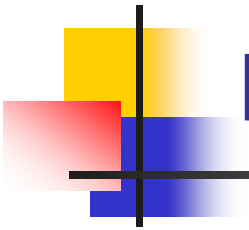


    2. If the node is the destination, then notify its upstream neighbor.



    3. Otherwise, notify both upstream and downstream neighbors.
- Backward link vice versa.



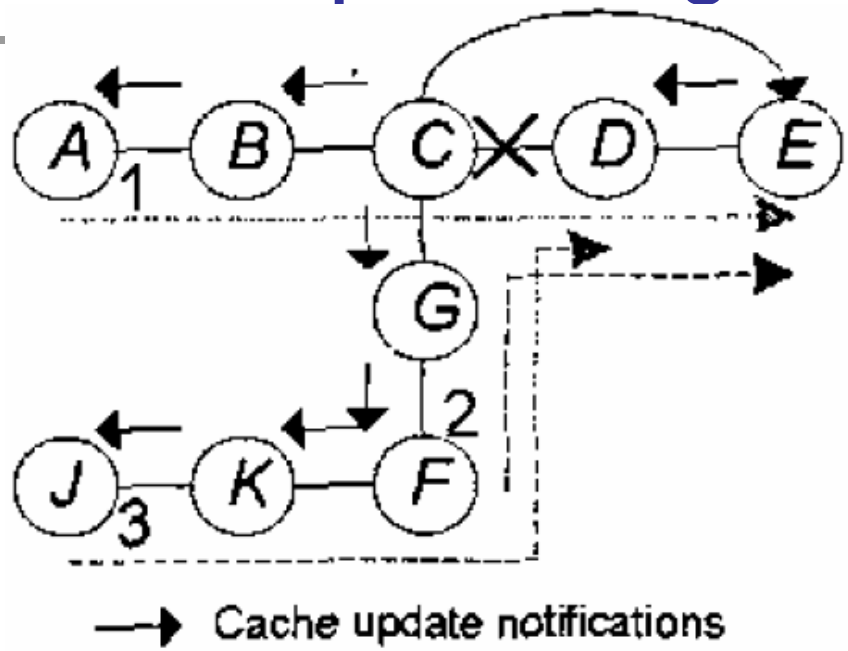


# Distributed Cache Update Algorithm

- After notifying the upstream and/or downstream neighbors, the algorithm checks the ReplyRecord field.
- If an entry contains a broken link, the algorithm notifies the neighbor that learned the link, then removes the table entry containing the broken link.

# Distributed Cache Update Algorithm

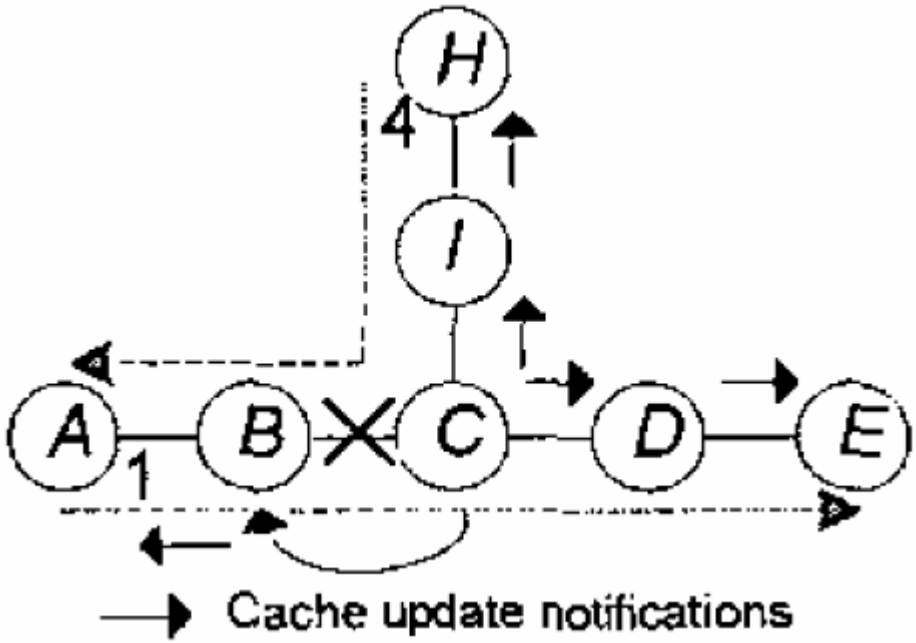
- Example 1




| | Route | S-D | DP | ReplyRecord |
|---|---|---|---|---|
| *C:* | *ABCDE* | *A E* | 2 | *G ← CDE* |
| *G:* | *GCDE* | *F E* | 0 | *F ← GCDE* |
| *F:* | *FGCDE* | *F E* | 0 | *K ← FGCD* |
| *K:* | *KFGCD* | *J D* | 0 | *J ← KFGCD* |
| *J:* | *JKFGCD* | *J D* | 0 | |

# Distributed Cache Update Algorithm

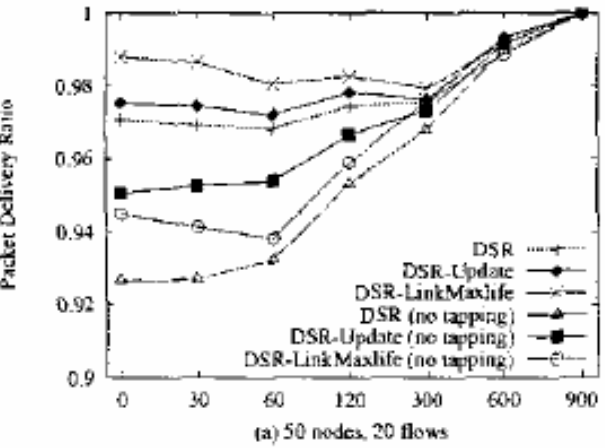
- Example 2

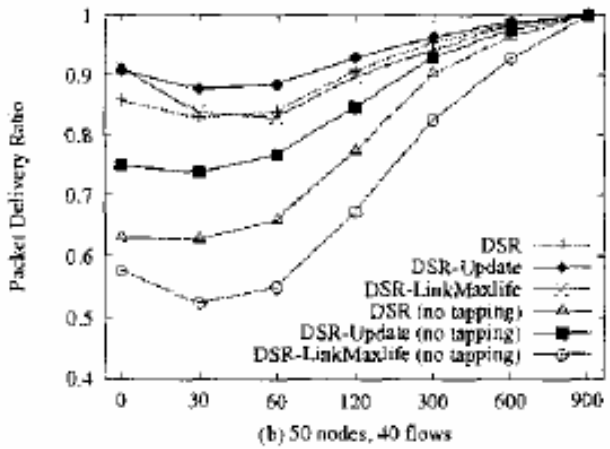


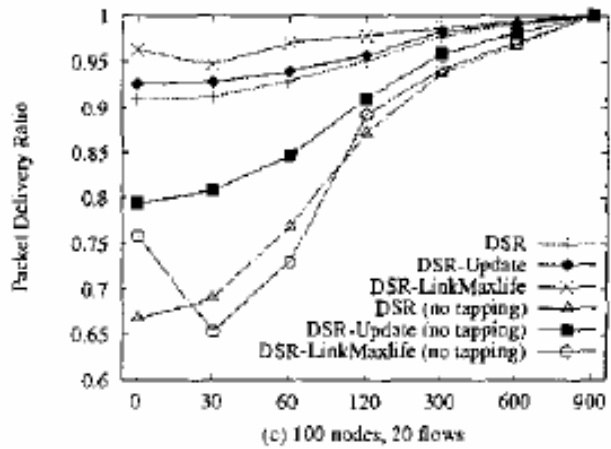

| | Route | S-D | DP |
|---|---|---|---|
| C: | *ABCDE* | *A E* | 2 |
| C: | *HICBA* | *H A* | 2 |

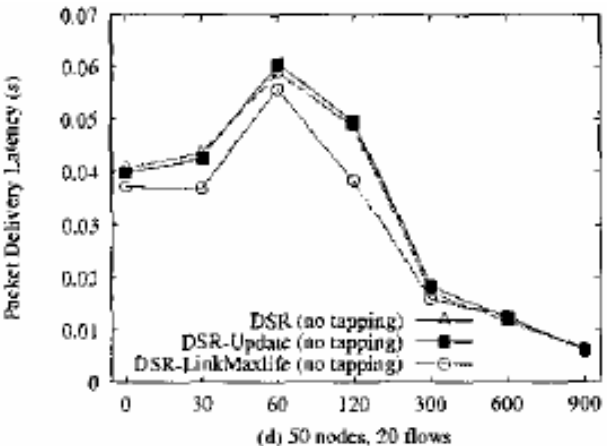
# Performance Evaluation

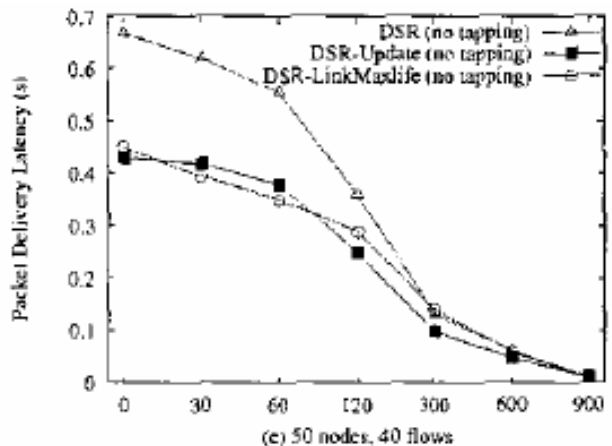
- Packet Delivery Ratio & Packet Delivery Latency

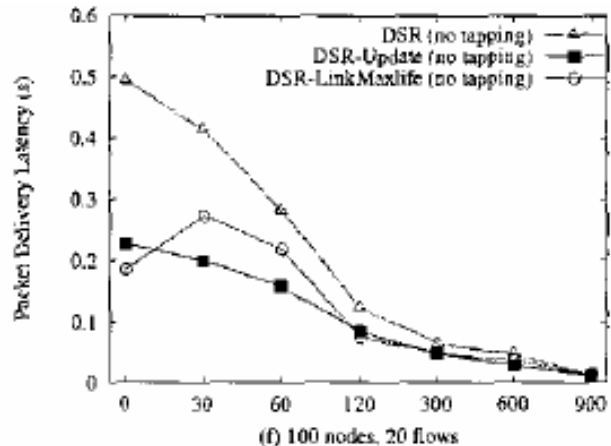


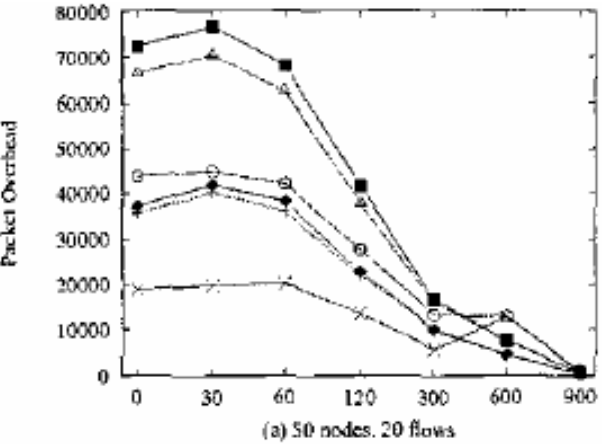



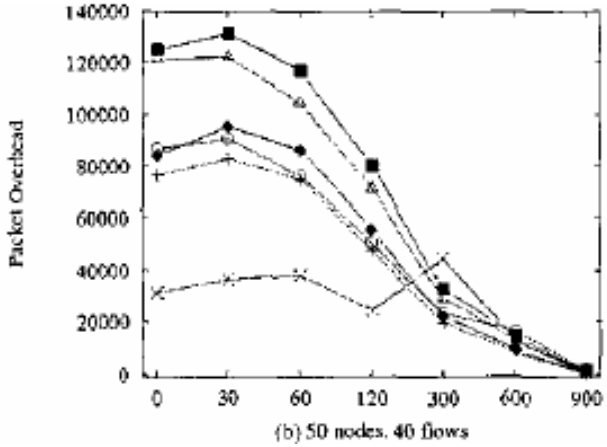



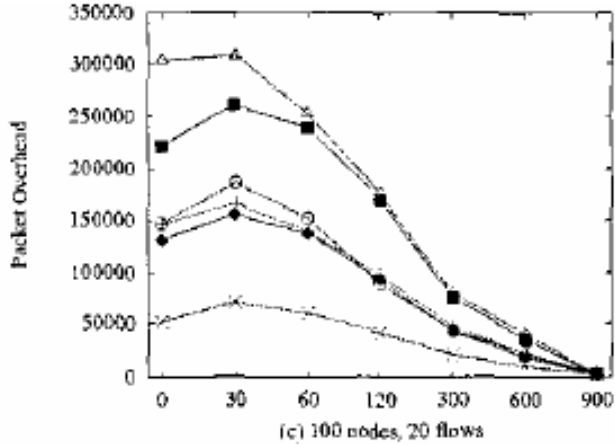



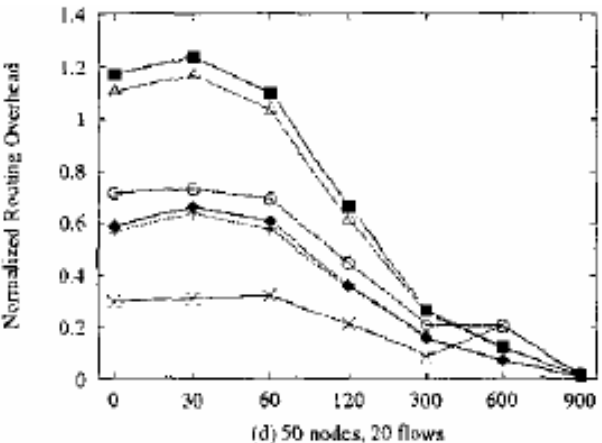
# Performance Evaluation

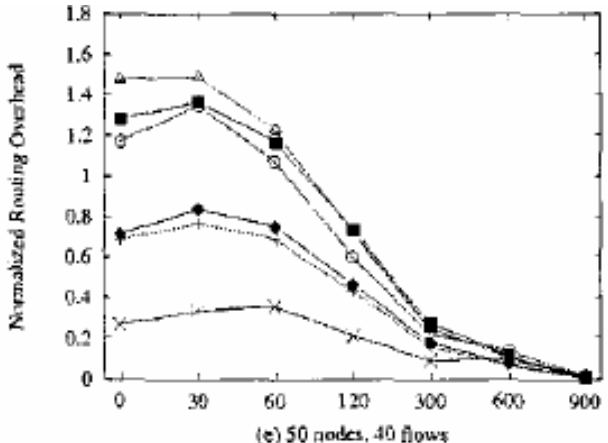
- Packet overhead & Normalized Routing Overhead

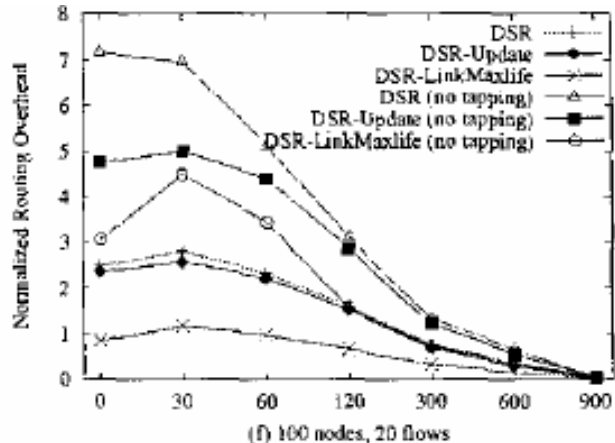


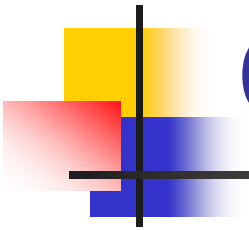





# Conclusions

- Authors defined a new cache structure called a cache table to maintain the information necessary for cache update.
- Based on the local information kept by each node, the distributed cache update algorithm disseminate the broken link information to all reachable nodes that have that link in their caches.
- Therefore, the algorithm enables DSR to adapt quickly to topology changes.