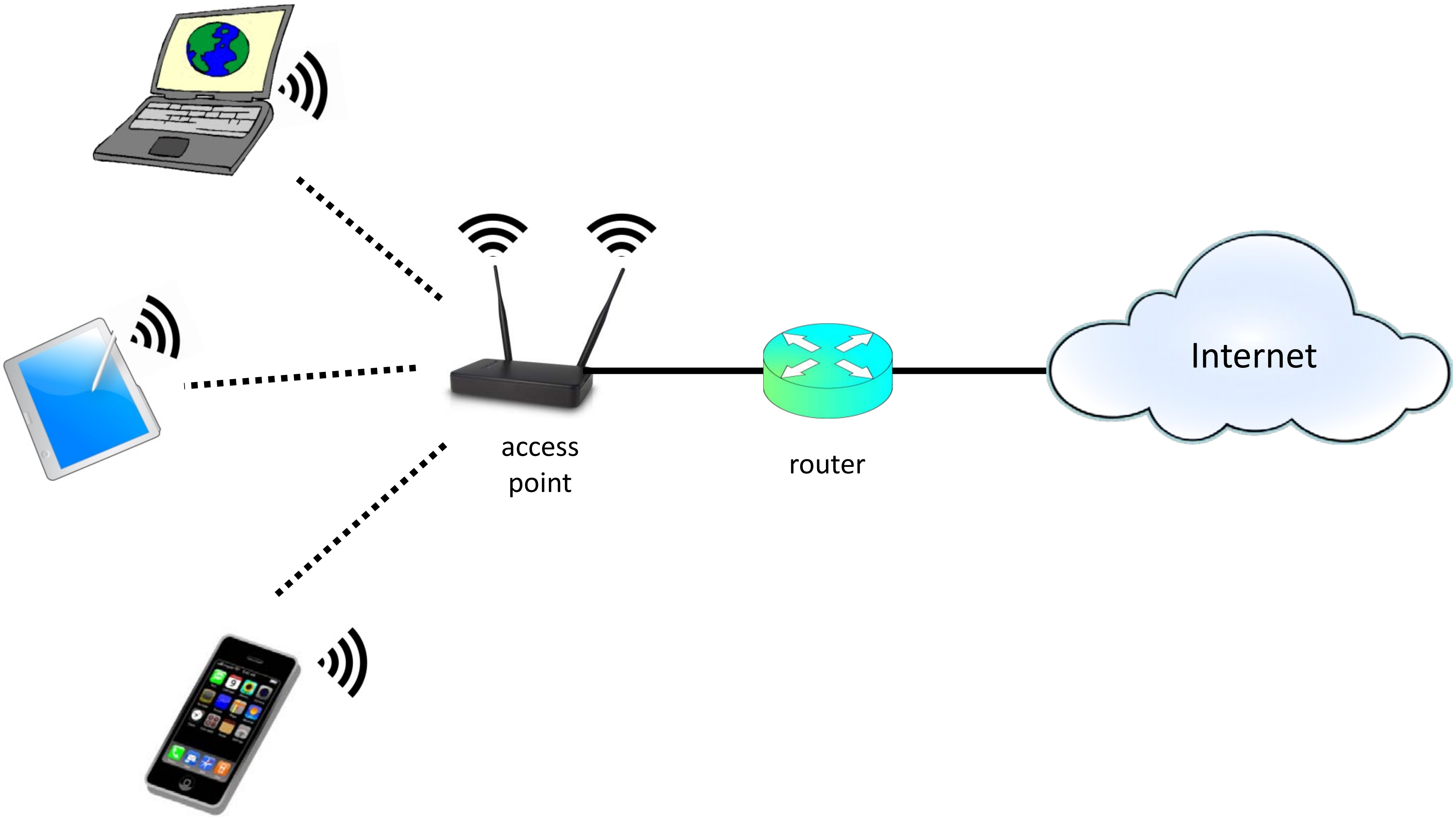
# On-Demand Routing in Wireless Ad-Hoc Networks with Wide Levels of Network Density

Presented by Wei-Cheng Xiao

Advisor: David B. Johnson

2015/03/30

# Wireless Network – Infrastructure Mode

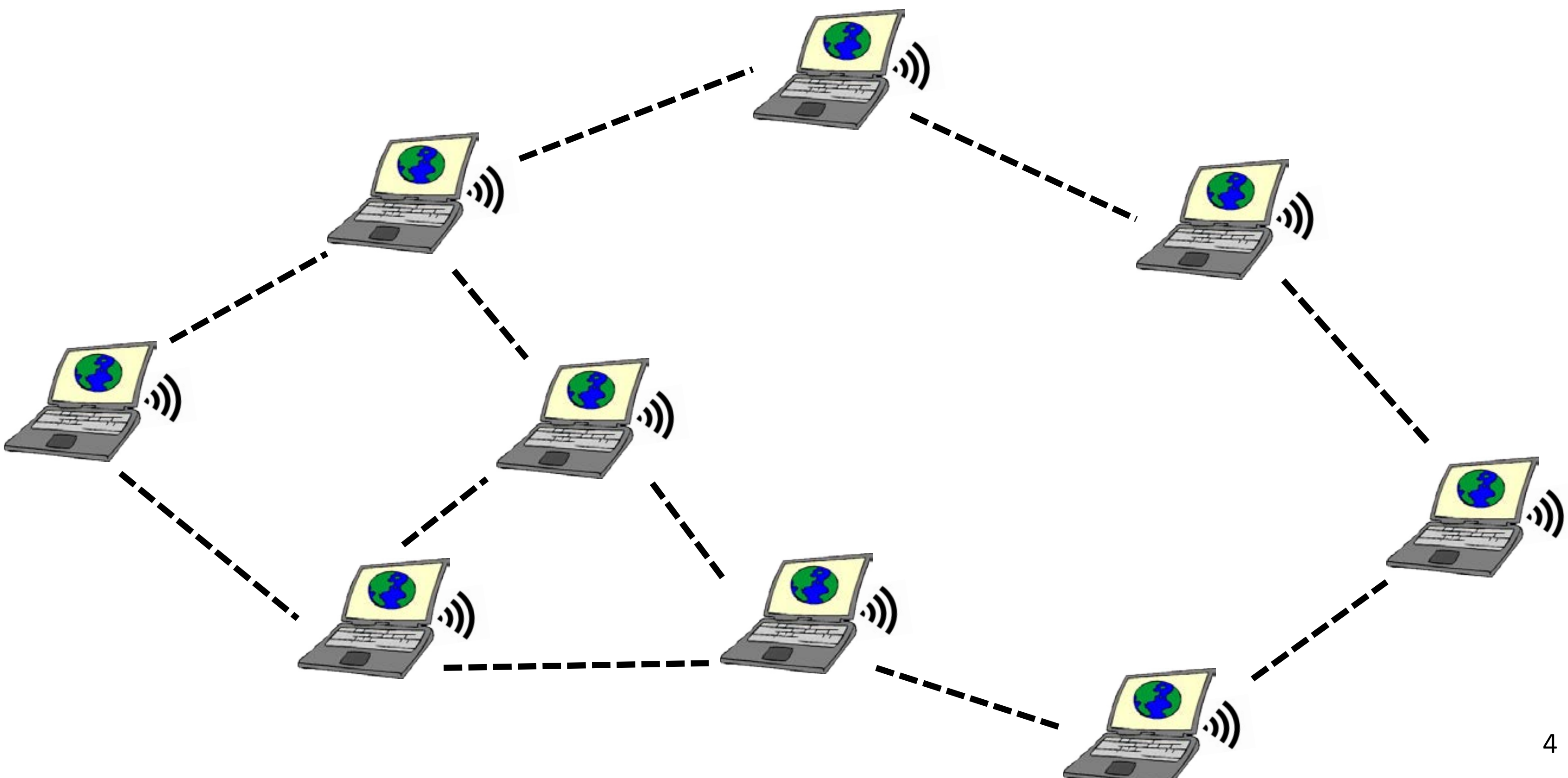


# When access point is not available…

- Computers still want to talk to each other.
  - Emergency situations
  - Search and rescue
  - Military purposes
  - Others…

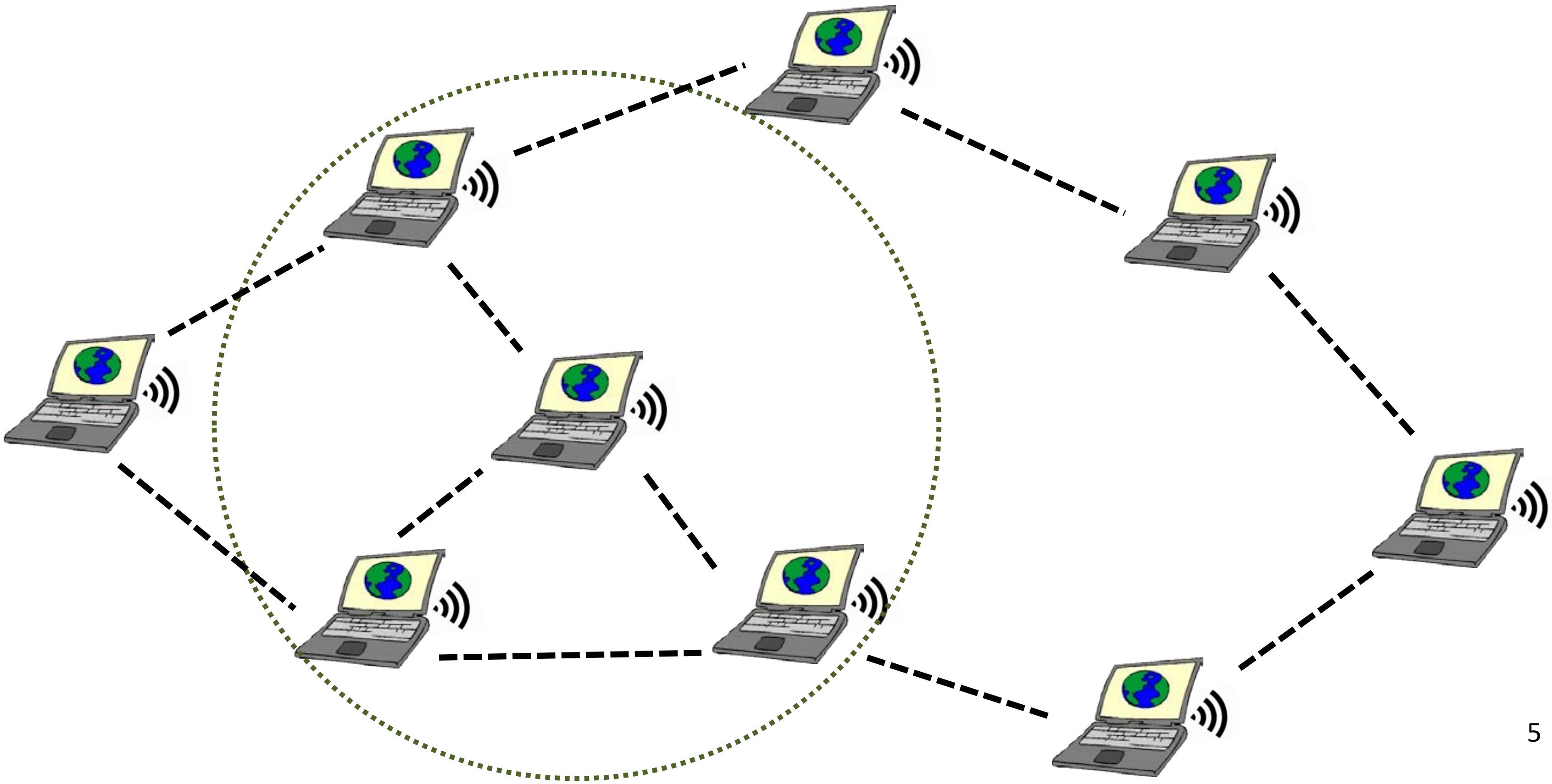
# Wireless Network – Ad-Hoc Mode

- No access point
- No centralized control
- Hop-by-hop message transmission

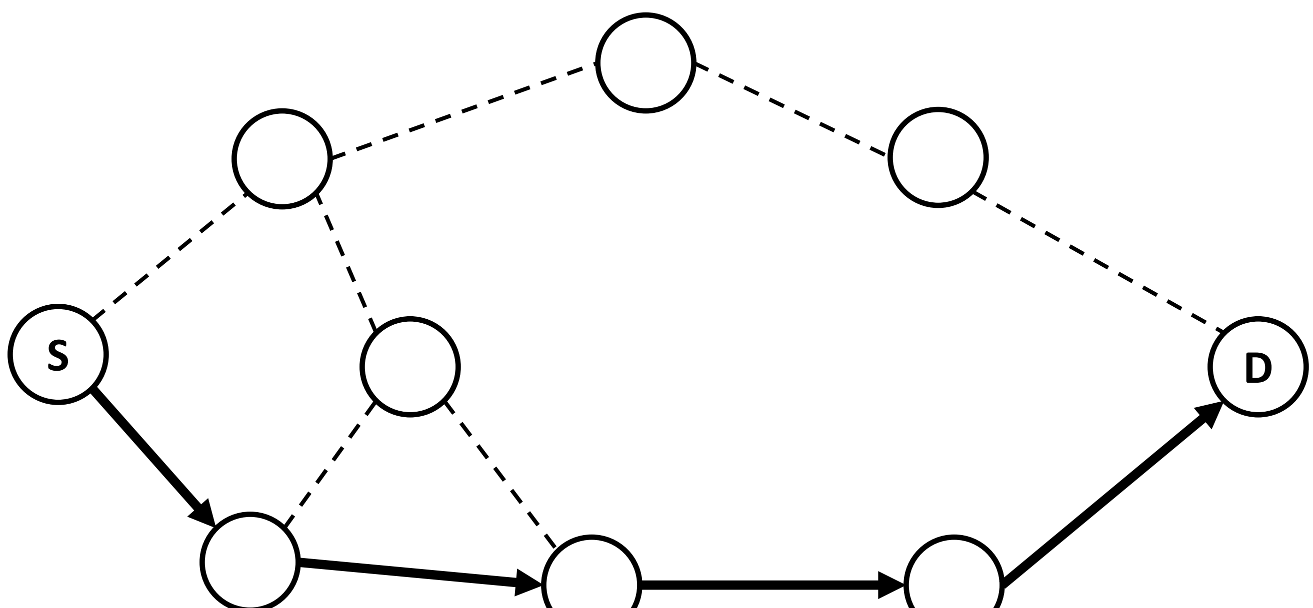


# Wireless Network – Ad-Hoc Mode

- No access point
- No centralized control
- Hop-by-hop message transmission

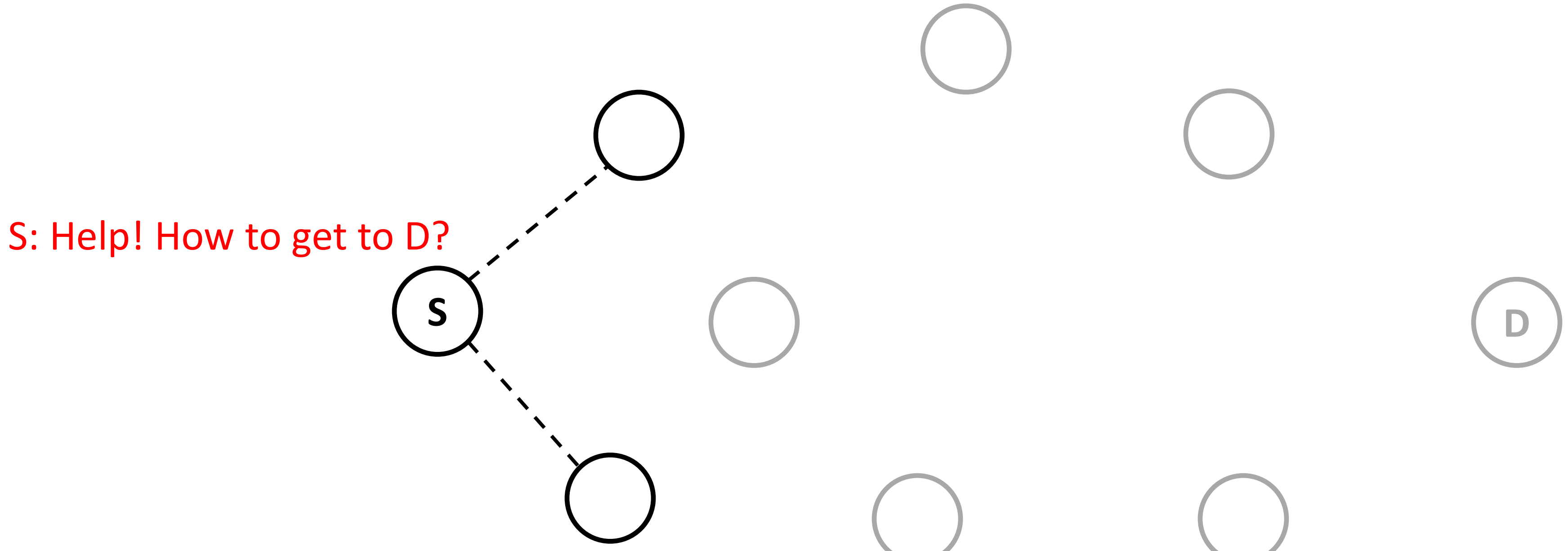


# Problems of Routing



# Problems of Routing

- Each node may only know its “neighbors”.
- The topology may change.
  - Nodes can be moving!



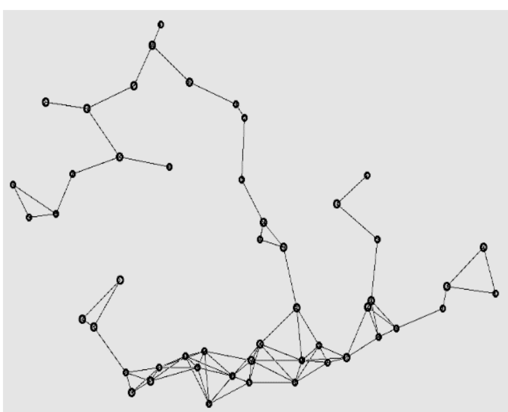
# Existing Solutions - Proactive

- **All** nodes **periodically** broadcast neighborhood info.
- Nodes learn about network by exchanging info with neighbors.
- E.g., link-state- or distance-vector- based routing protocols.
- Disadvantages:
  - high overhead, even in a static network,
  - possible routing loops.

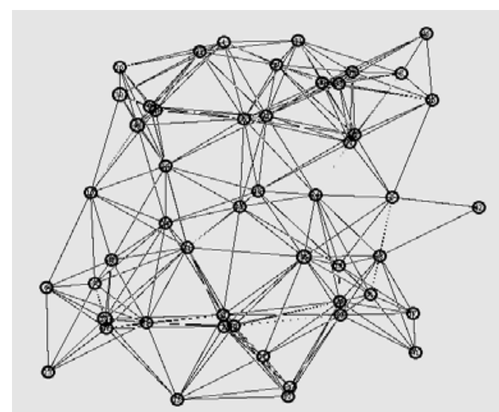
# Existing Solutions – On-Demand

- Each node performs route discovery only when necessary.
- No topology change → no extra overhead.
- E.g., DSR, DYMO.

- Problem:
  - None of them has been studied in **sparse** networks.



sparse vs. dense



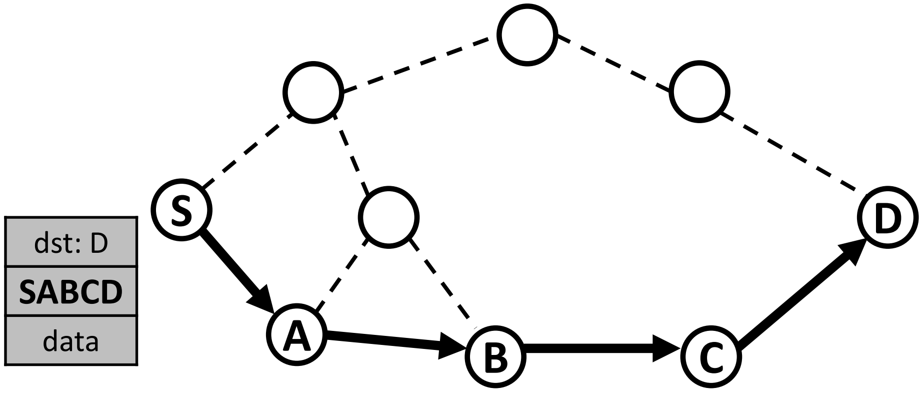
# Our Work

- Choose Dynamic Source Routing (DSR) as the routing protocol for study.
- Consider the drawbacks of DSR in sparse networks.
- Extend the design of DSR to improve its performance in sparse networks.
- Evaluate DSR with new design in both dense and sparse networks.

# Background of DSR

# Background of DSR

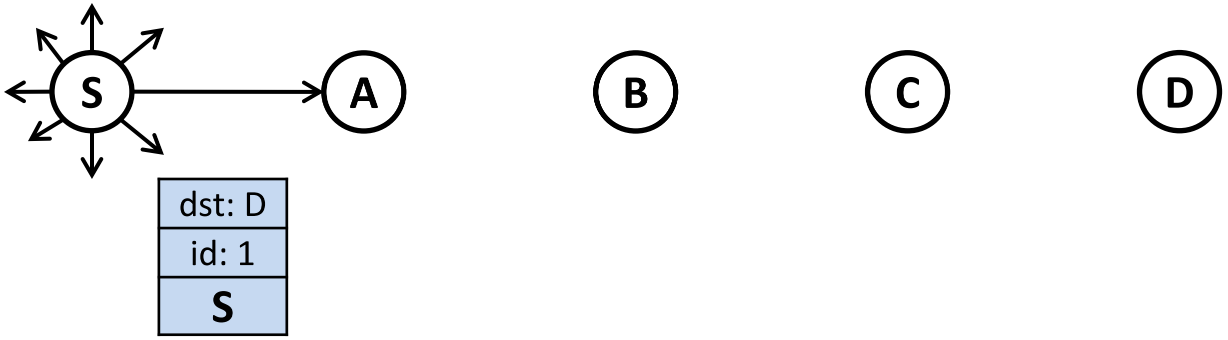
- Source routing for every unicast packet
  - Source determines the route for a packet to the destination.
    - The route is stored in the packet header.




- DSR control packets
  - For route discovery and route maintenance
    - Route Request
    - Route Reply
    - Route Error

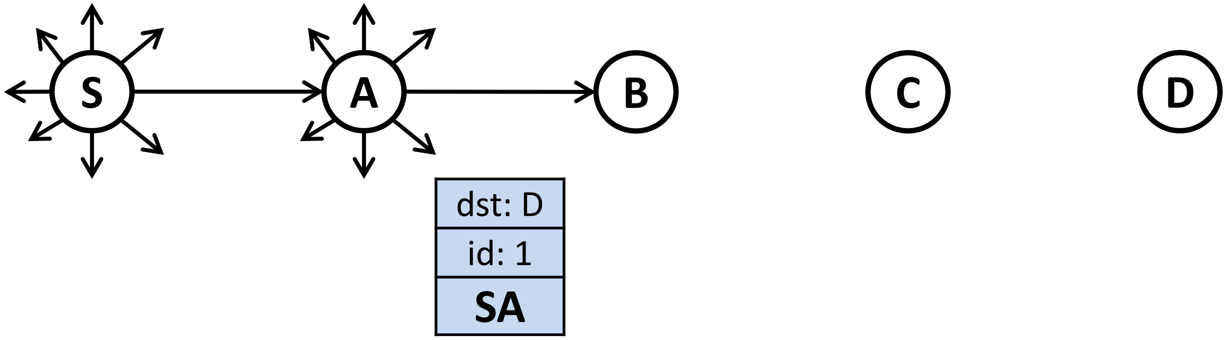
# Route Discovery

- To find out a route:
  - Source floods a Route Request into the network.
    - Purely on-demand.
  - A Route Request contains a unique ID.
    - Each node broadcasts the same Route Request only once.



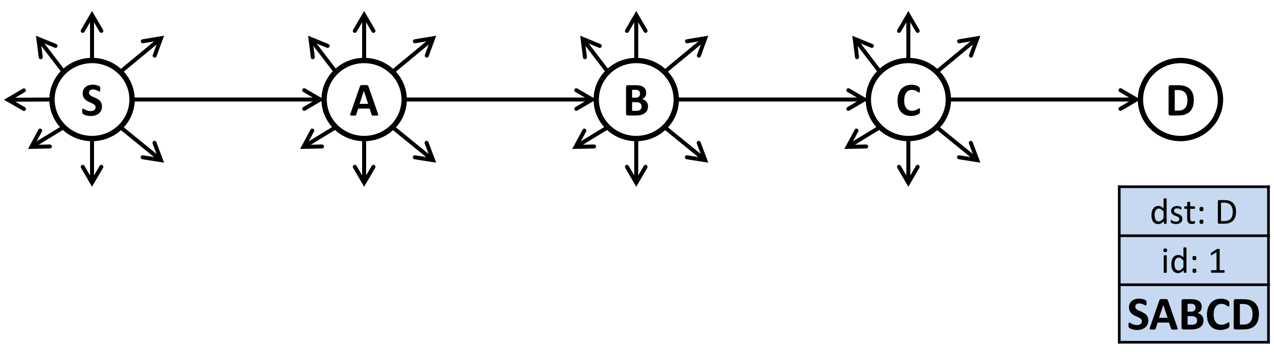
# Route Discovery

- To find out a route:
  - Source floods a Route Request into the network.
    - Purely on-demand.
  - A Route Request contains a unique ID.
    - Each node only broadcasts the same Route Request once.
  - Node appends its own address when forwarding the Route Request.



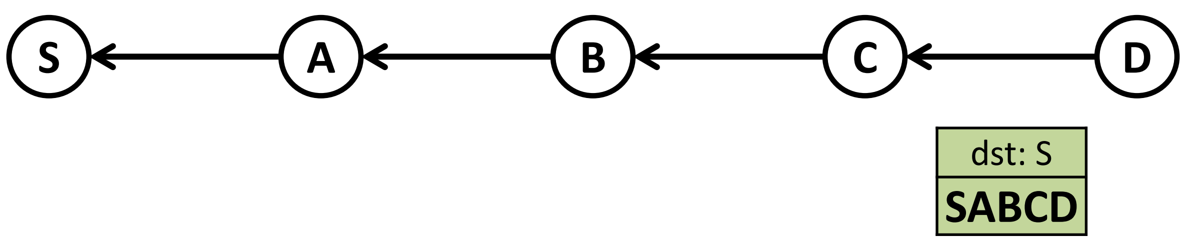
# Route Discovery

- To find out a route:
  - Source floods a Route Request into the network.
    - Purely on-demand.
  - A Route Request contains a unique ID.
    - Each node only broadcasts the same Route Request once.
  - Node appends its own address when forwarding the Route Request.



# Route Reply

- The destination sends a Route Reply back to the source using unicast to tell the source the route.
  - One Route Reply for **every** Route Request received.
    - The source may finally learn multiple routes to the destination.
    - The source chooses the “best” route.



# Route Cache

- Route discovery is expensive.

- To reduce overhead
  - Each node remembers links it has learned from the source route header
    - From packets it received,
    - From packets it overheard.

| |
|---|
| dst: X |
| ABCD |
| … |

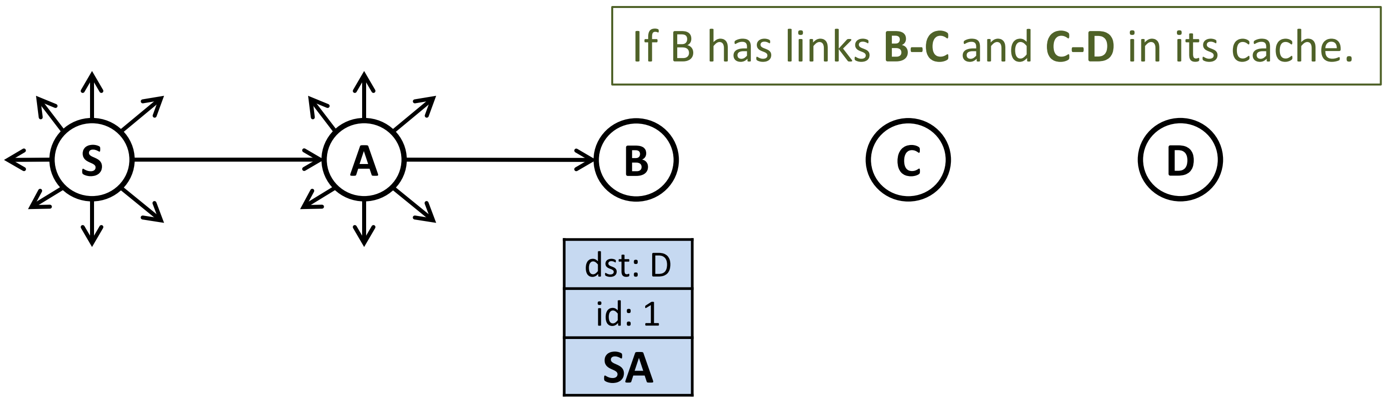
(caches links **A-B**, **B-C**, and **C-D** )

  - For each node, cached links form a subgraph of the network.
  - When looking for a route, search the cache using Dijkstra's algorithm before doing route discovery.

# Reply from Cache

- A node replies for a Route Request directly if it already knows a route to the destination.
  - Also stops forwarding the Route Request.

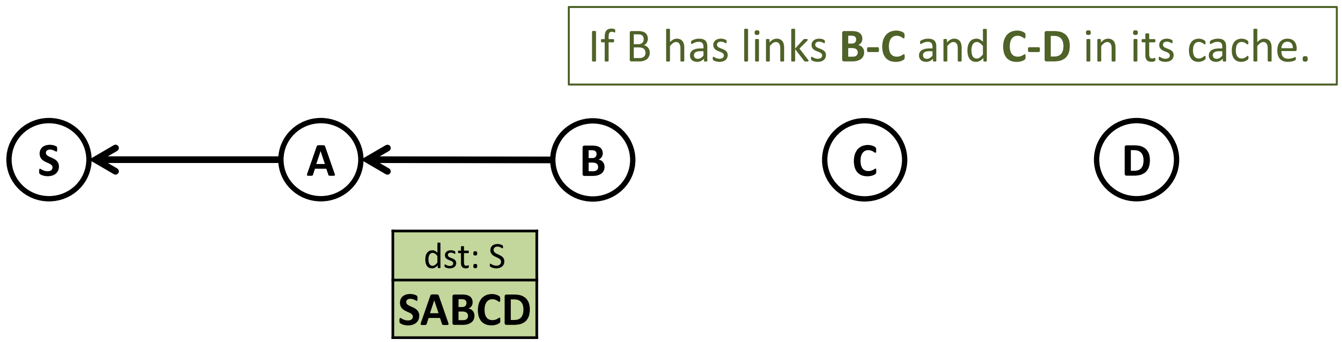
- Further reduces overhead from route discovery.



# Reply from Cache

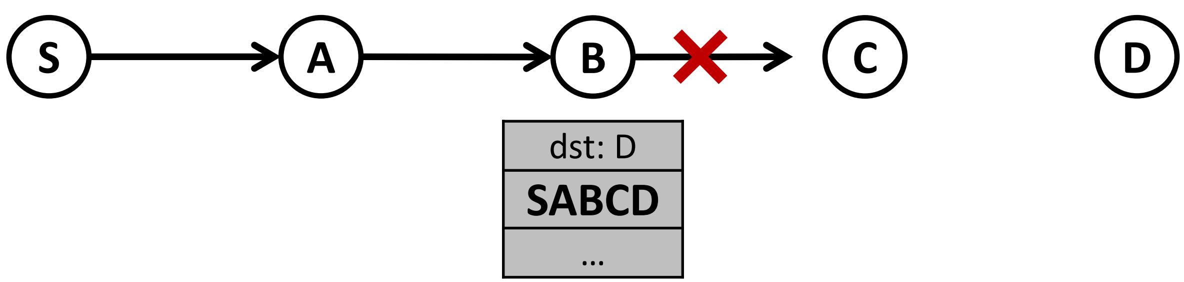
- A node replies for a Route Request directly if it already knows a route to the destination.
  - Also stops forwarding the Route Request.

- Further reduces overhead from route discovery.



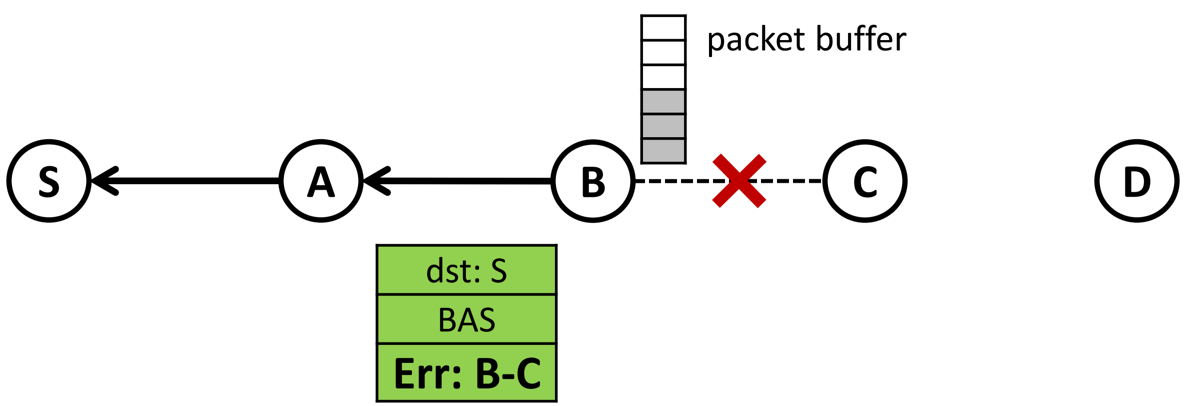
# Broken Links

- Some links in the source route header may no longer exist.
  - E.g., due to node movement.
- A link is considered broken if no ACK is received.
- To deal with broken links
  - Route Error
  - Salvaging



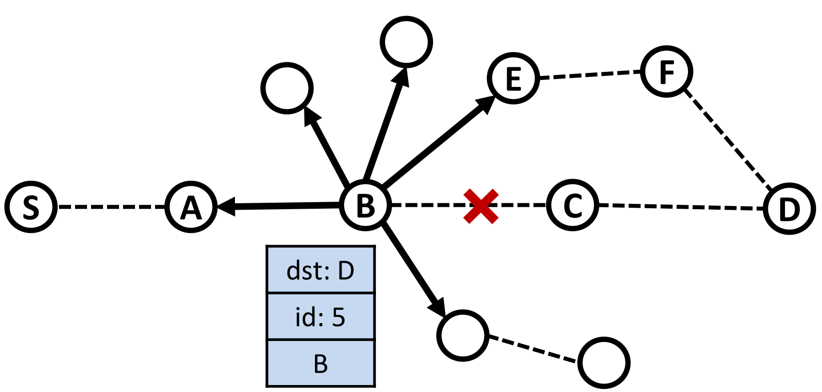
# Route Error

- When a broken link is detected, a Route Error is sent back.
  - Nodes which receive/overhear the Route Error remove the broken link from cache.
  - Prevents nodes from using/telling others the same broken link.
- The original data packet is then buffered, waiting for being salvaged.



# Salvaging

- The salvager first checks its own cache.
- If no alternative route is found, the salvager then performs a 1-hop route discovery.
  - In order not to create too much overhead.

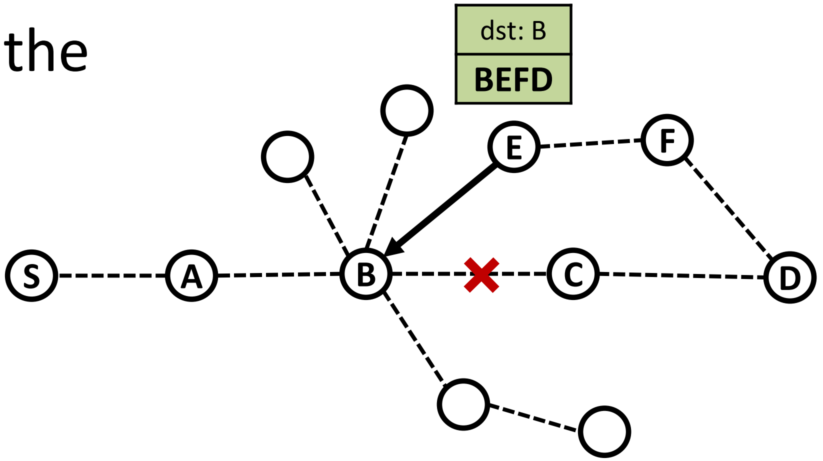


# Salvaging

- The salvager first checks its own cache.
- If no alternative route is found, the salvager then performs a 1-hop route discovery.
  - In order not to create too much overhead

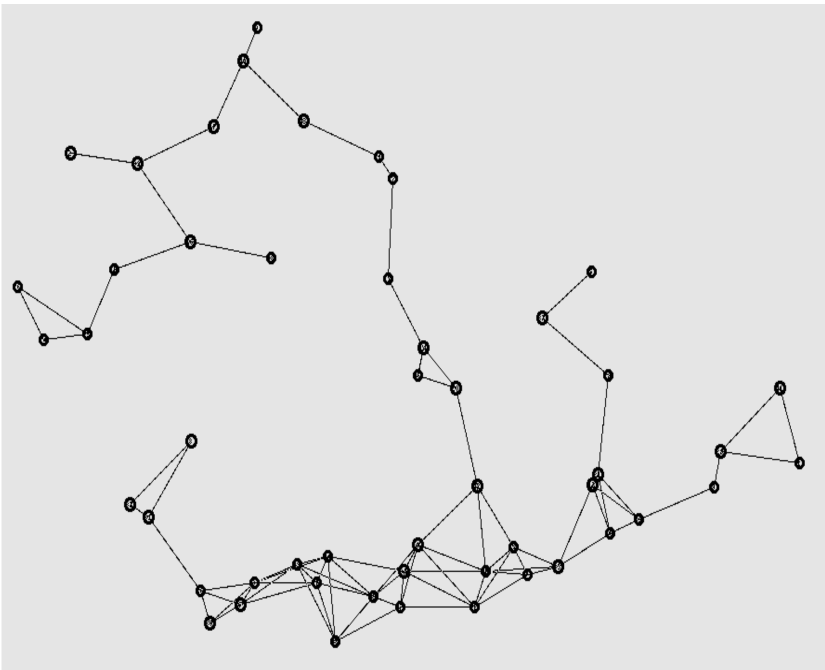
- Neighbors which know a route to the destination will reply from cache.

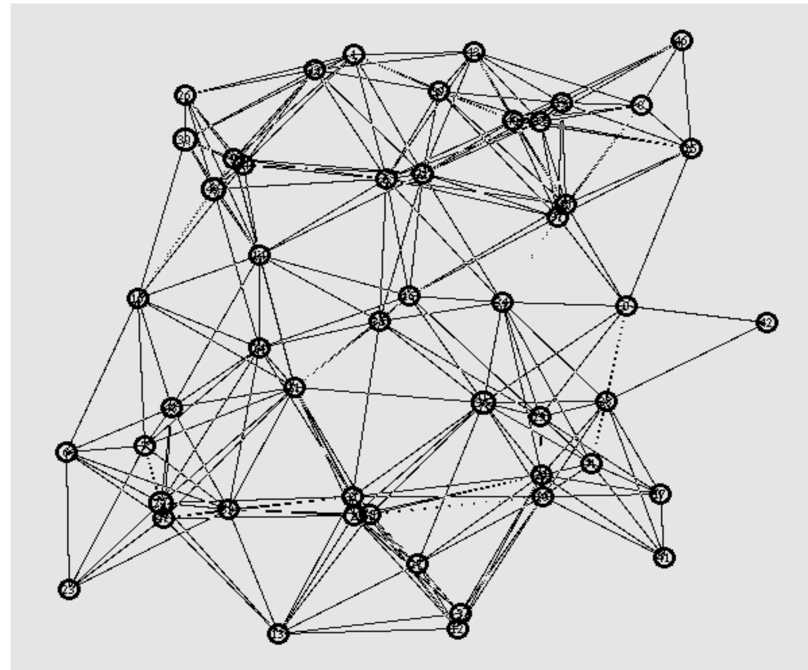
- If no Route Reply is received, the salvager will try again later.



# DSR in Sparse Networks

# Sparse vs. Dense





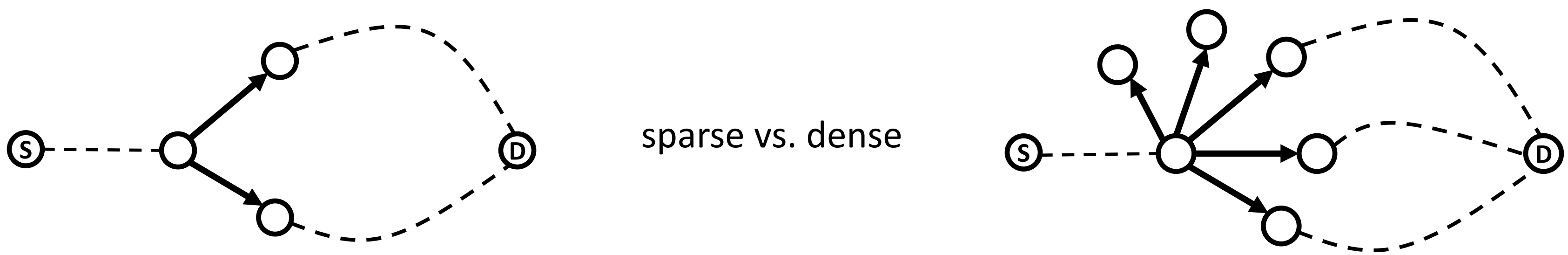
- Characteristics of sparse networks
  - Fewer route options
  - More shared links among flows



(The sparse mobility model was designed and implemented by Keyvan Amiri.)

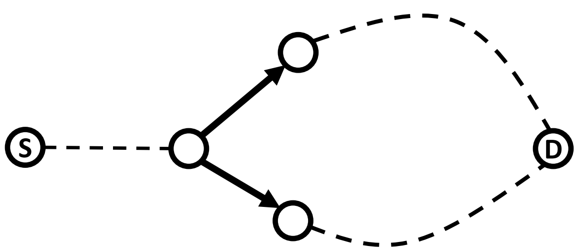
# Characteristic 1 – Fewer Route Options

- Brings problems to salvaging in DSR.
  - Fewer neighbors → a smaller chance to find alternative routes.

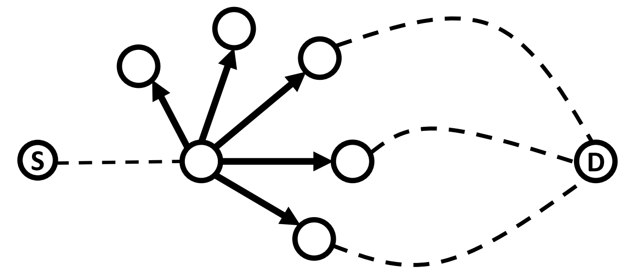


# Characteristic 1 – Fewer Route Options

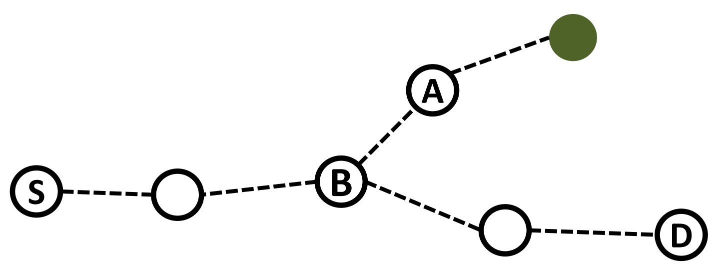
- Brings problems to salvaging in DSR.
  - Fewer neighbors → a smaller chance to find alternative routes.




sparse vs. dense

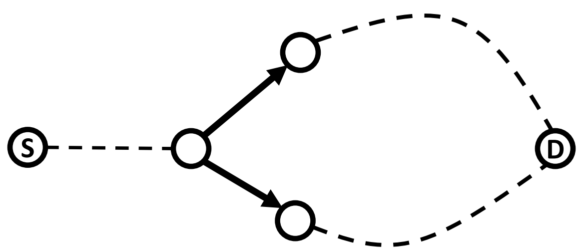



  - The salvager may be located at a "dead end" due to node movement.

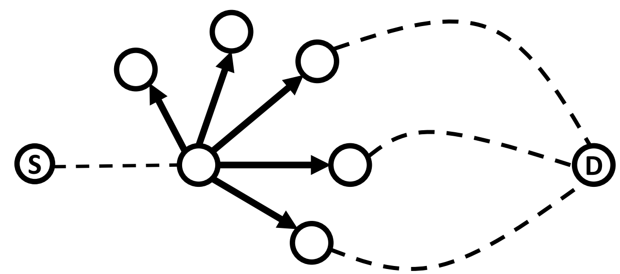


# Characteristic 1 – Fewer Route Options

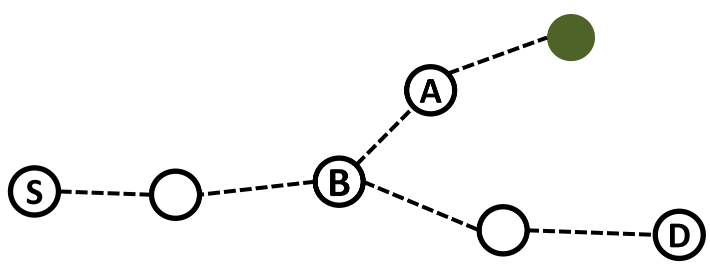
- Brings problems to salvaging in DSR.
  - Fewer neighbors → a smaller chance to find alternative routes.




sparse vs. dense



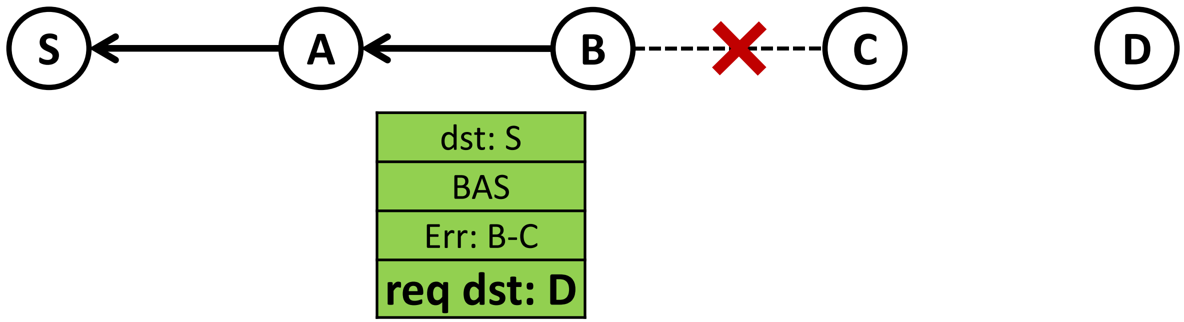

  - The salvager may be located at a “dead end” due to node movement.




  - In original DSR, the salvager keeps trying 1-hop route discovery periodically until it is “lucky”.
    - Long network latency

# Current Partial Solution

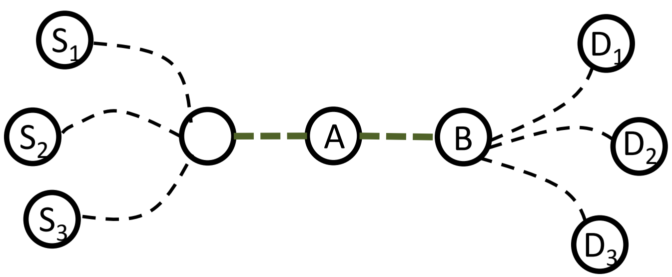
- Embed the request to destination in the Route Error.



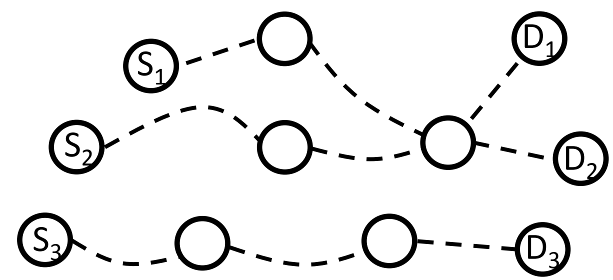

- Send Route Error all the way back to the source.
  - If anyone in the middle knows an alternative route to the destination, send a Route Reply to the salvager.
    - Mitigate the “dead end” problem.
  - Otherwise, the source will do a network-wide route discovery.
    - Mitigate the problem in general cases.

# Characteristic 2 – Shared Links

- In sparse networks, it is more likely to have multiple shared links among different flows.

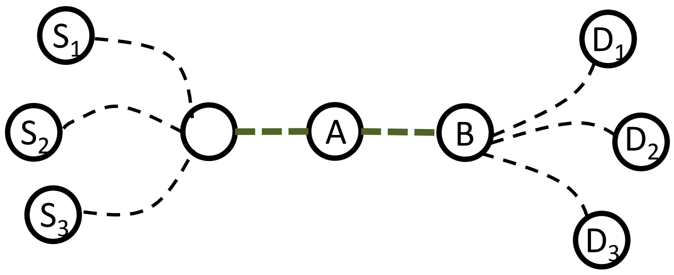



sparse vs. dense

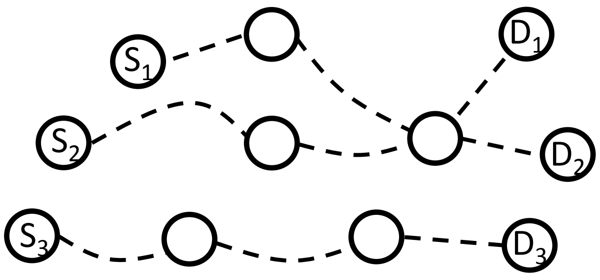


# Characteristic 2 – Shared Links

- In sparse networks, it is more likely to have multiple shared links among different flows.




sparse vs. dense




- Once a shared link is broken, multiple flows will be affected.
- In original DSR, Route Error is triggered by packet
  - Better to tell other sources earlier.

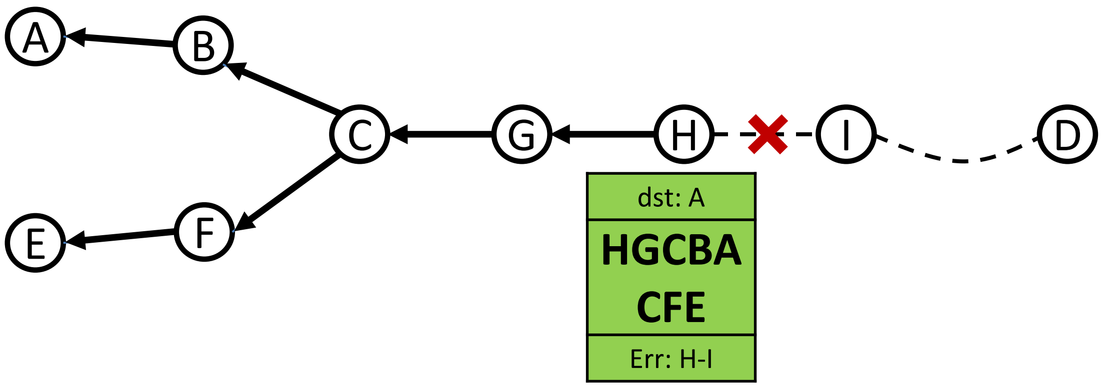
# Solution: Route Error Multicast

- Send one single Route Error to all the source nodes that have recently used the broken link.
  - In current implementation, recently = in the previous 3 seconds.
  - For each outgoing link of a node, the node remembers a list of sources that have recently used the link.

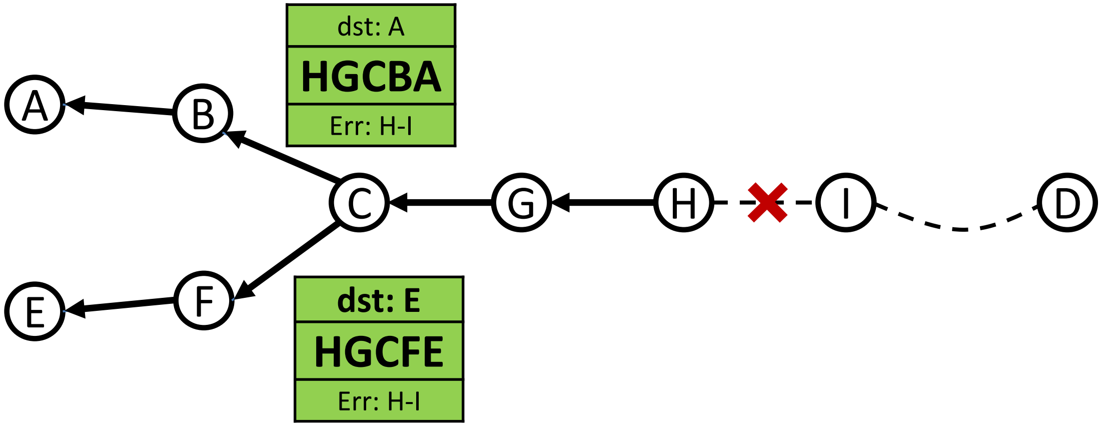
- Encode a multicast tree in the source route header.

# Solution: Route Error Multicast

- The tree is represented as branches listed in DFS order.




- The Route Error is divided into multiple packets at the branch point.



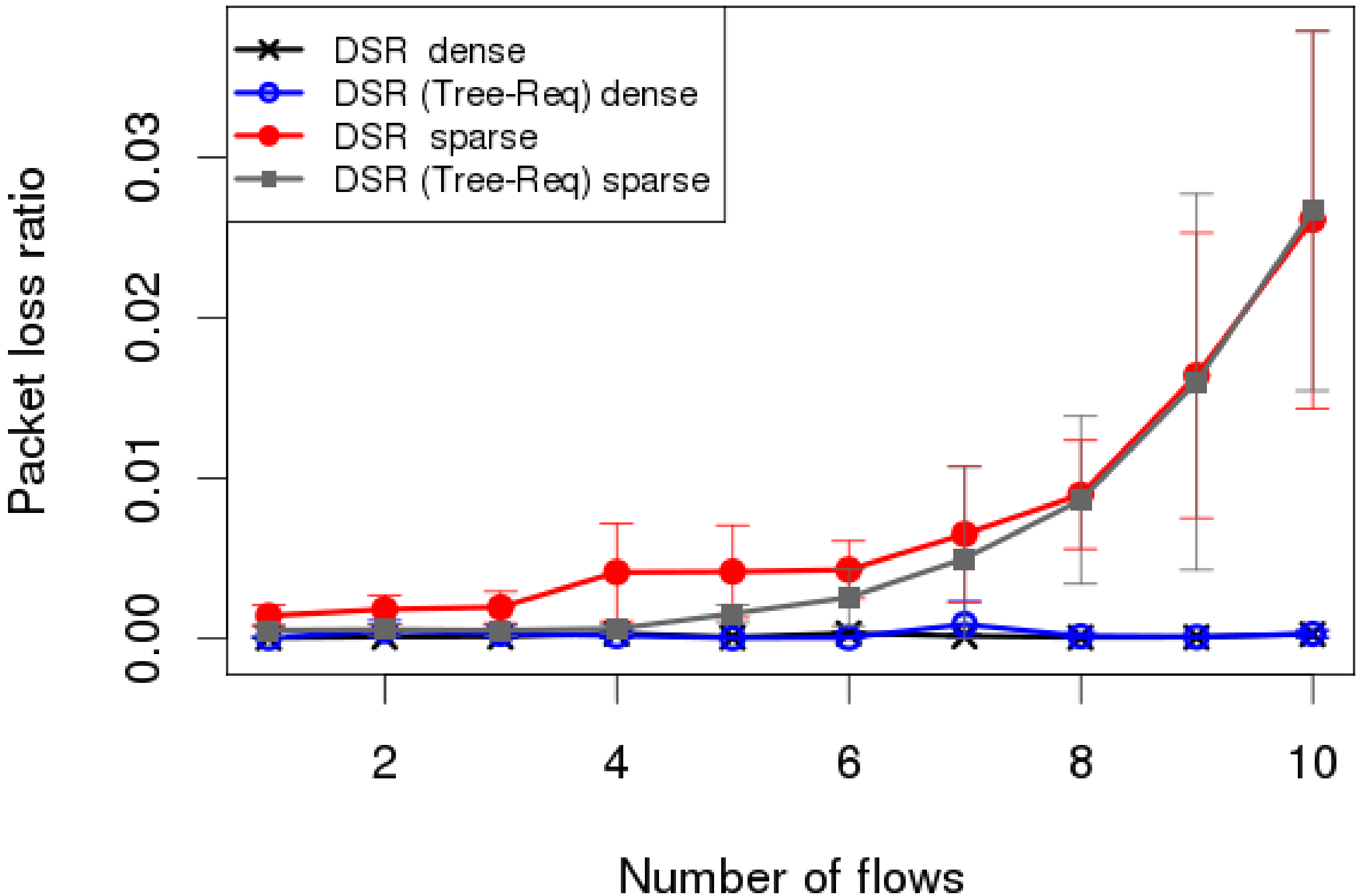

- This can be combined with the previous "request to destination" feature.

# Performance Evaluation

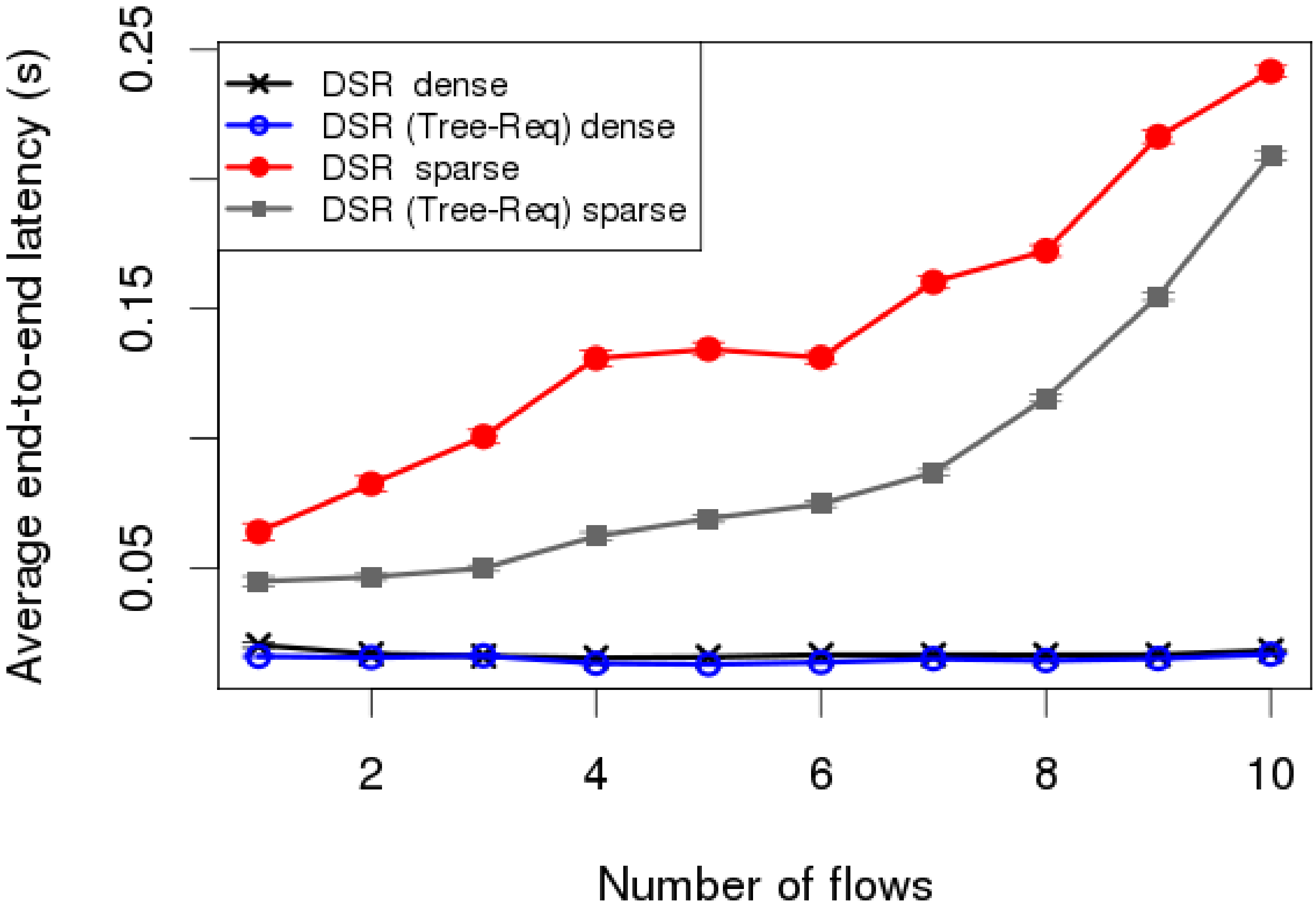
# Simulation Setup

- Use the ns-2 simulator.
- Parameters
  - 50 nodes, each with TX range 250 m
  - 1 ~ 10 flows
  - 10 dense and 10 sparse network topologies
    - Node speed: 5 ~ 20 m/s
  - Data traffic: 10 packet/s generated periodically from the application
  - Each simulation runs for 900 seconds (simulated time).
- Facts
  - Average route length
    - Dense: 2.7 hops
    - Sparse: 9.5 hops
- Assumptions
  - All links are bidirectional.
  - No network partition.

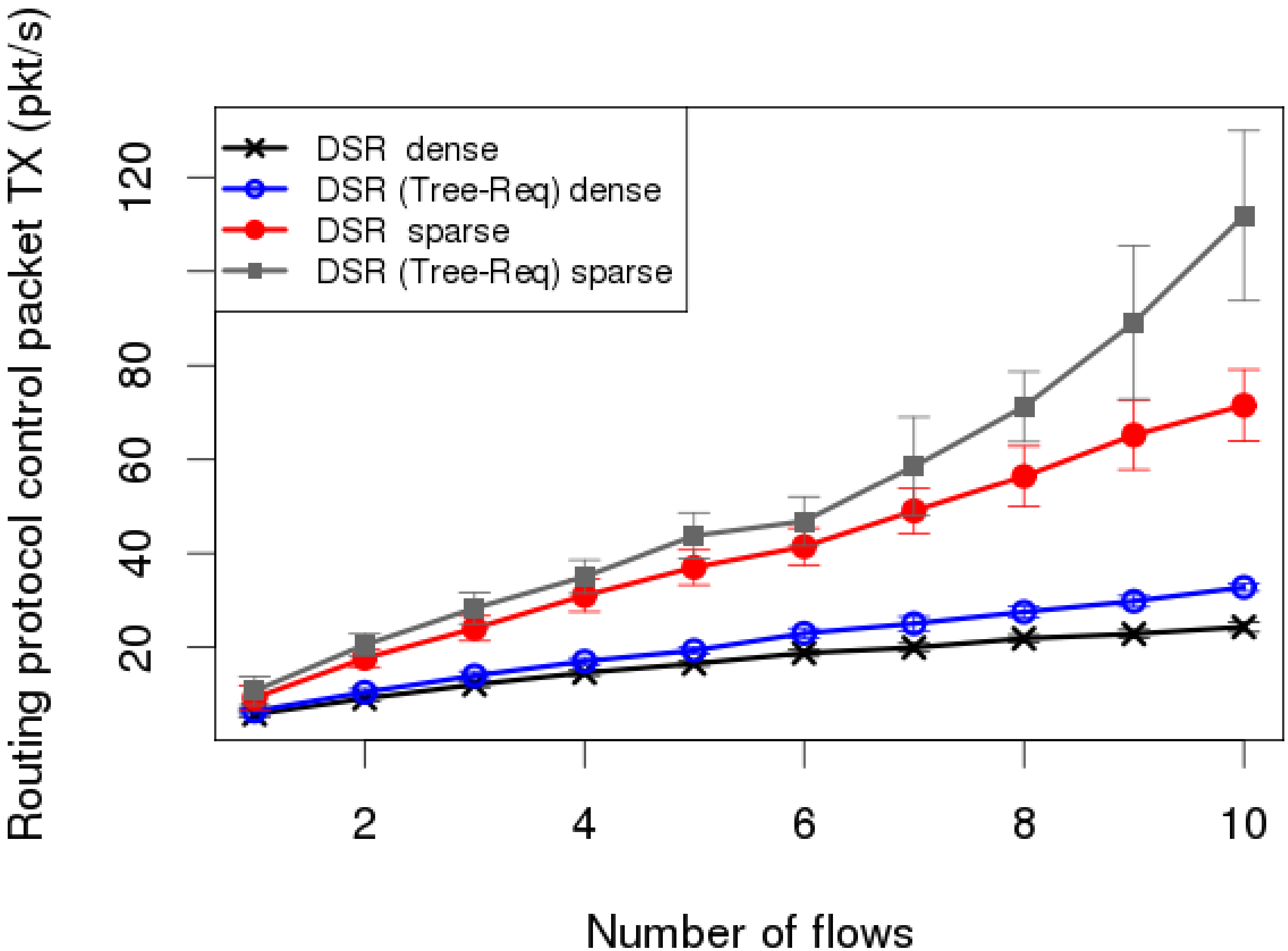
# Packet Loss Ratio



# Average End-to-End Latency



# Control Packet Overhead



# What is next?

- Reduce the overhead of Route Error
  - Change per-packet Route Error to periodic Route Error
- Reduce the overhead of Route Request
  - Merge route discoveries for multiple destinations into a single route discovery.

# What is next?

- Reduce the overhead of Route Error
  - Change per-packet Route Error to periodic Route Error
- Reduce the overhead of Route Request
  - Merge route discoveries for multiple destinations into a single route discovery.

- Increase the successful rate of salvaging (reduce the time spent on salvaging)
  - When embedding the “request to destination”, embed multiple addresses in the suffix route