Technische Universität Dresden

Chair for Telecommunications
Lehrstuhl Telekommunikation

Dimitri Marandin, Jose Gonzalez

www.ifn.et.tu-dresden.de

Link Cache Validation Mechanism for Dynamic Source Routing (DSR) in Ad hoc Networks

1. Abstract

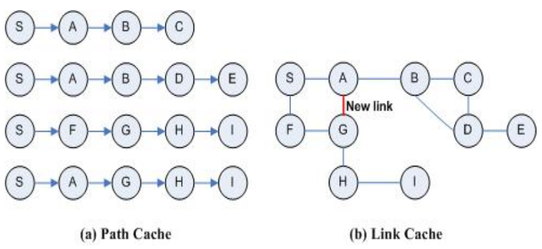
An *ad hoc network* is a collection of mobile hosts with wireless network interfaces that may form a temporary network without the aid of any established infrastructure or centralized administration. In such environment, the nodes operate both as hosts as well as routers. Due to mobility, the topology of the network may change randomly, rapidly and unexpectedly. Because of these aspects and the fact that the resources are limited in mobile nodes, an efficient routing in ad hoc networks is a crucial and challenging problem.

2. Introduction

Dynamic Source Routing (DSR) is one of efficient on-demand routing protocols. The main idea of on-demand routing is to find and maintain *only needed routes*. The obvious advantage of discovering routes on-demand is to avoid overhead costs of maintaining routes that are not used.

In order to avoid the need for such a route discovery to be performed before each data packet is sent, such routing protocols must cache previously discovered routes.

There are two types of cache structure.



(a) Path Cache (b) Link Cache

3. Problem Statement

Due to mobility, cached routes easily become stale. The adverse effects of stale routes can be summarized as:

- **Causing packet losses, increasing packet delivery latency and routing overhead.**
- **Degrading TCP performance.**
- **Wasting the energy of source nodes and intermediate nodes.**

4. Linkcache Validation Mechanism

Our approach (DSR-DistributedFailureInform) maintains a topology propagation state in a distributed way. In a link cache a node stores additional information for each cached link: which node's neighbour has learned which links from this node.

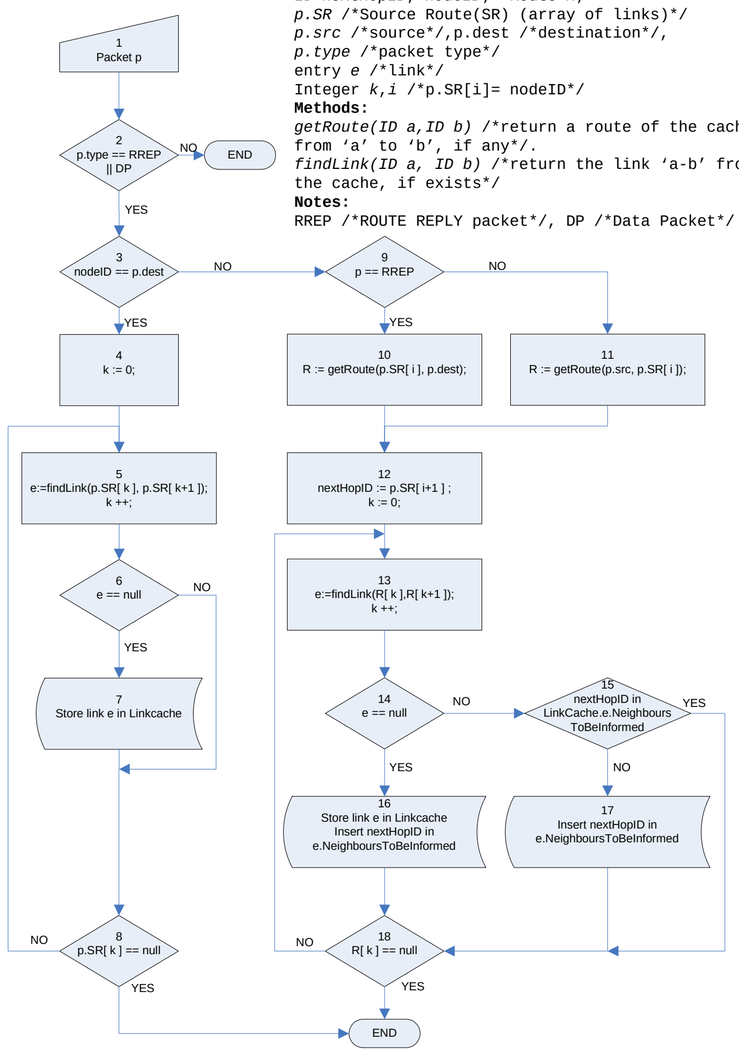
Each node gathers this information during route discoveries and data transmission. To remove stale routes, such information is sufficient in most cases because each node knows for each cached link which neighbours have that link in their caches.

DSR-DFI has three main procedures:

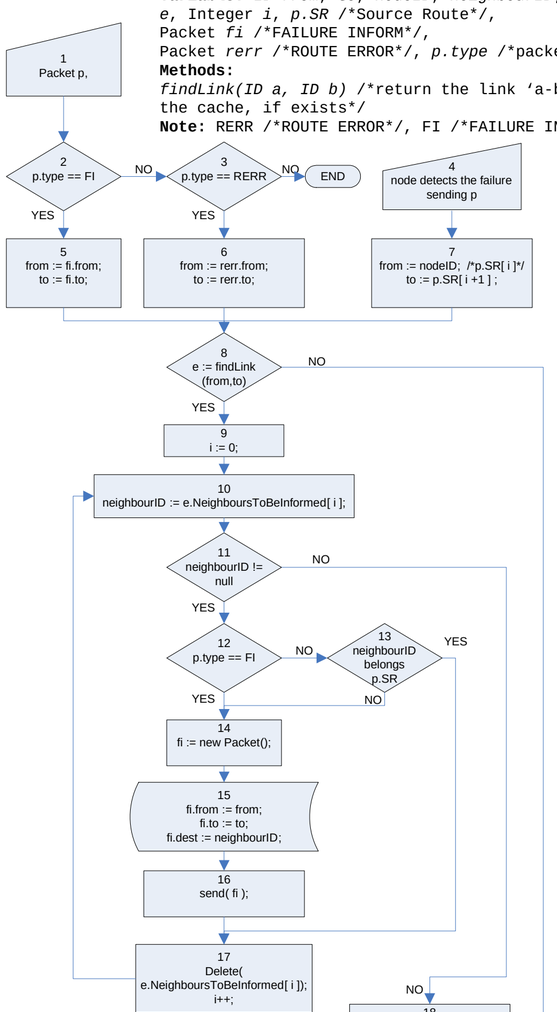
addRoute – to insert the links together with information necessary for updates.

findRoute – to search for routes maintained in the cache.

linkcacheValidation – to disseminate link failure information through the network and remove stale links .



Procedure: addRoute
Input: Packet *p*
Variables:
ID *nextHopID*, *nodeID*; Route *R*;
p.SR /*Source Route(SR) (array of links)*/
p.src /*source*/,p.dest /*destination*/,
p.type /*packet type*/
entry *e* /*link*/
Integer *k,i* /*p.SR[i]= nodeID*/
Methods:
getRoute(ID a,ID b) /*return a route of the cache from 'a' to 'b', if any*/.
findLink(ID a, ID b) /*return the link 'a-b' from the cache, if exists*/
Notes:
RREP /*ROUTE REPLY packet*/, DP /*Data Packet*/



Procedure: linkcacheValidation
Input: Packet *p*
Variable: ID *from*, *to*, *nodeID*, *neighbourID*; entry *e*, Integer *i*, *p.SR* /*Source Route*/,
Packet *fi* /*FAILURE INFORM*/,
Packet *rerr* /*ROUTE ERROR*/, *p.type* /*packet type*/
Methods:
findLink(ID a, ID b) /*return the link 'a-b' from the cache, if exists*/
Note: RERR /*ROUTE ERROR*/, FI /*FAILURE INFORM*/

We have made a new DSR packet called FAILURE INFORM

IP-Header	packet type	from	to

Field *packet type* indicates that the packet is a FAILURE INFORM and the information about the broken link is contained in the next two fields.

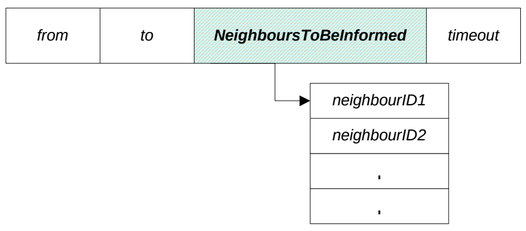
The fields *from* and *to* indicate the end-points of the broken link .

There are two possible inputs in the procedure *linkcacheValidation*:

1) a node receives a packet informing about the error (ROUTE ERROR or FAILURE INFORM packet),

2) a node detects a link breakage when it attempts to deliver a packet to the next hop of the source route.

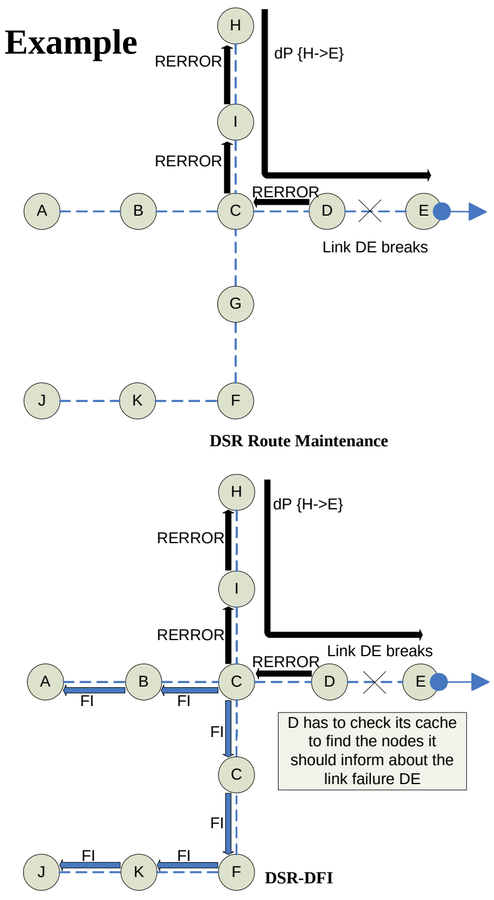
New field in link cache entry: *NeighboursToBeInformed*. It represents a list of neighbourIDs, which have to be informed in case of failure in the link represented by the fields *from,to*.



All the links maintained in link cache are bi-directional.

The timeout additionally used is *Link-Adapt*. Each node keeps a table recording its perceived stability of each other node. When a link is used, the stability metric for both endpoint nodes is incremented. When a link is observed to break, the stability metric for both endpoints is multiplicatively decreased.

5. Example

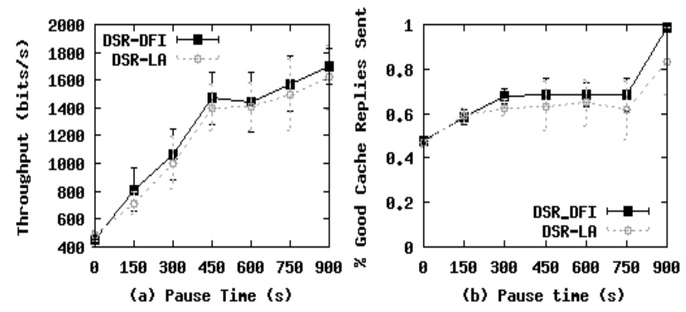


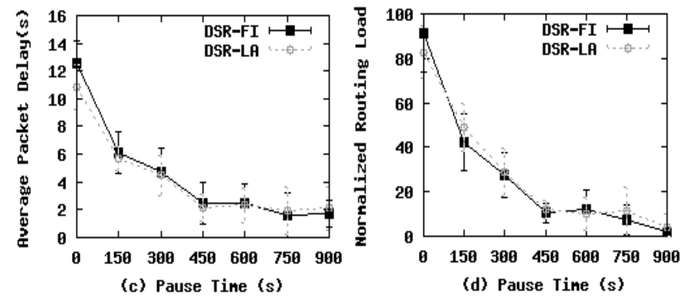
6. Performance Evaluation

The simulation model used:

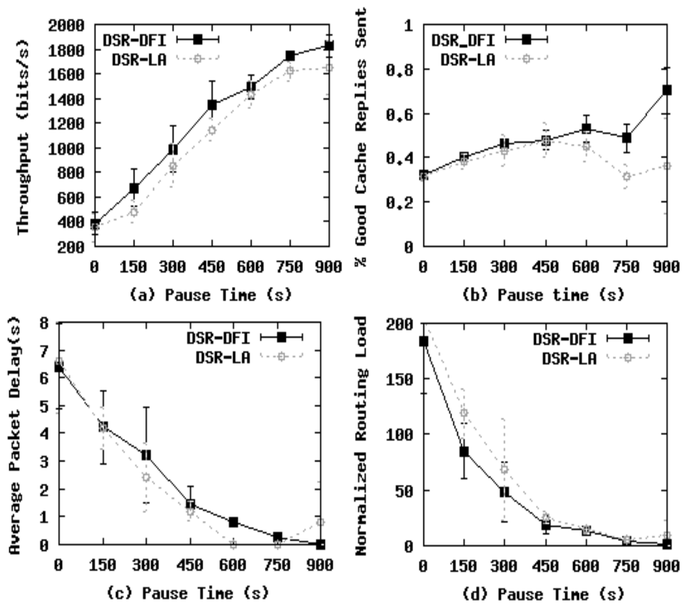
- Simulator: ns-2.
- Number of nodes: 50, 100.
- Traffic model: CBR, 4pkts/s, 64 bytes.
- Number of simultaneous flows: 10, 20, 30, 40.
- Mobility model:
  "steady-state" Random Waypoint model.
  Velocity: Uniform distributed [1,19] m/s.
  Pause time: 0, 150, 300, 450, 600, 750, 900 s.
- Simulation areas: 1500 m x 500 m, 1500 m x 1500 m.
- Simulation time: 900 s.

We compared our approach (DSR-DFI) with DSR with Link-Adapt timeout (DSR-LA).





1500 m x 500 m, with 50 nodes and 10 flows.



1500 m x 1500 m, with 100 nodes and 10 flows.

7. Conclusions

Our approach (DSR-DistributedFailureInform) informs the nodes of the network about link breakages in a distributed way. It allows to update link caches quickly and with minimum overhead. Simulative investigation has shown better performance of our approach against timeout-based invalidation algorithms. Especially good results have been obtained for delivery ratio and throughput.