Routing in wireless networks

In MANETs nodes organize and maintain the network by themselves and they can be both router and hosts; the routing problem consist of *find and maintain* a route, provided that it exist. How to ensure that a route exist is another problem than routing; nodes are assumed to be cooperative.

Issues

. Self-organized network implies having a distributed algorithm.

. Topologies change dynamically over time due to mobility and time-varying channel nature.

. Link failure and repair.

. Asymmetric links.

. Limited bandwidth: protocols that use flooding create a lot of overhead.

. New performance criteria like route stability despite mobility, energy consumption, available bandwidth, number of hops that implies constraint on the delay.

Protocols and approaches

Based on the following distinction, it is possible distinguish:

. topology based routing protocols: DSDV, OLSR, DSR, AODV;

. position based routing protocols.

Approaches can be:

. preventive (proactive): are mechanisms based on distance vector and link-state, similar to traditional routing problem in wired network; they try to maintain consistent and fresh information by propagating updates; examples are DSDV, OLSR;

. reactive: routes are created when desired by the source node; two phases are important: route discovery and route maintenance; examples are DSR and AODV.

Preventive approaches implies low latency (for this reason are suitable for environment with time and delay constraints) but high overhead (due to routing tables), while, at the contrary, reactive approaches ensure low overhead, but high latency, therefore are more suitable for networks with limited bandwidth.

Preventive protocols

DVR

Distance vector routing discovers routes, but does not provide maintenance and no a priori knowledge of the network is required. Each node has a routing table that describe distances to all reachable destinations: this information is propagated to all neighbors to inform them of the distances. Initially each node has infinite cost to reach all other stations and 0 to reach himself: after information on the routing tables are sent and received, the node can build his own routing table; the algorithm converge by iteration.

Issues

Distance vector does not scale well with the number of nodes, the complexity is $O(n^2)$; the convergence and the recovery time are slow and it may suffer the count-to-infinite problem.

Destination Sequenced Distance Vector

DSDV is based on Bellman-Ford routing approach and each node maintains routing tables that reports all possible destinations; each entry contains:

. destination node;

. next hop;

. number of hops to the destination;

. a sequence number assigned by the destination: it is used to distinguish old and new routes and allows to avoid loops.

Routing tables are periodically transmitted to maintain consistency and are possible:

. full dumps to report the entire routing tables;

. incremental updates that describe just changes performed since last full dump.

When a route update is received and the node does not have that information, it adds an entry to his table; in the other case, it checks the sequence number and updates the table if the information is fresh because just fresh routes are used (if two routes have the same sequence number, the actual route used is the one with the smaller metric). A failure of a link implies putting an infinite cost to that link: this feature and sequence numbers allows to deal with count-to-infinite problem. The link failure is detected by checking for how long time a node does not receive any message from his neighbors or if the link layer fails to deliver packets. Nodes, moreover, have to keep track of the settling time of routes and fluctuations.

Advantages and Disadvantages

a) Moderate memory requirement.

a) Guarantees loop-free paths.

a) Simple update coordination protocol.

d) Provide a single path between a given pair source-destination.

d) Selection of the periodic update interval and the time before assume that a route is considered *old* (stale).

d) Update overhead.

OLSR

OLSR (Optimized Link State Routing) forecast that sources uses MPRs to build routes: MPRs, Multipoint Relays are only nodes that can forward link state updates; normal station does not do that. In this way, OLSR allows to improve the efficiency of flooding by minimizing useless retransmissions, therefore is very effective for dense networks, while in sparse networks it reduces to be a pure link state approach.

The election of MPRs happens like: each node select as MPR his one hop neighbors that allows to reach his two-hops neighbors. If a link is unidirectional between A and B, B is considered as a 2-hop neighbor reachable through C, for example.

How the protocol works

Periodically each node has to send *Hello* messages to his 1-hop neighbors every Hello interval; each message contains:

. his id;

. the list of his 1-hop neighbors.

Through Hello message the node is able to build the *Neighbor Table*: this table includes all 1-hop and 2-hop neighbors of a given node; with neighbor tables nodes can elect their MPRs because there is also information about the type of links (bidirectional or unidirectional).

MPRs send *Topology Control* (TC) messages every TC period: they are used to advertise link states and contain the list of 1-hop neighbors who have selected the considered node as MPR. For example:

$$[A - B]$$

can be a TC message where:

. A is the selected destination;

. B is his MPR, the last node to cross before reaching A.

Through TC messages is possible build the Topology table; each entry includes:

. the address of the destination;

. the address of his MPR;

. the MPR selectors set sequence number.

Each node is in charge to maintain a *Routing table* where entries record all possible destinations; routing tables are based on Neighbor table and Topology tables. Each entry includes:

. the distance (in number of hops);

. the address of the possible destination;

. the address of the next hop to reach the destination.

Routing tables are build in the following way:

. first set the distance $h = 1$ and take all 1-hop neighbors from the Neighbor table (NOT include neighbors reachable through unidirectional links);

. then, from topology table select with $h = 2$ possible destinations reachable through nodes already present in the routing table;

. iterate following the same procedure and, at each step increase h.

Observations

Since Hello and TC messages are sent periodically, no acknowledges are necessary: this allows also to consider the case in which transmissions are not reliable. Of course, each message contains a sequence number to detect outdated information.

The overhead depends in principal from all interval message periods: short implies very high overhead but short time for reaction.

The peculiar feature is the choice of MPRs: performances depends on this. Indeed, a right selection allows to provide a very effective flooding. Those nodes are the ones that expire the more traffic load and, consequently, they consume more energy that other nodes. Another issue is mobility: MPRs mobility is very critical because if a route breaks ensure reliability is difficult.

The last issue is the typology of links accepted: only symmetric links can be used.

Reactive protocols

Reactive protocols look for a route just when it is needed. Protocols analysed are:

. DSR;

. AODV.

DSR

DSR (Dynamic Source Routing) is composed by two phases:

. route discover;

. route maintenance.

Moreover, DSR, uses the *soft state* approach and source routing; through the soft state approach, the entire state is discovered only when needed and, in case of failure, can be easily re-discovered without affect protocol's performances. The *source routing* is a technique for which the source has to specify in the header the all path and not only the next hop. Due to this fact, the network can not be too large, otherwise, if the distance of the farthest node is too large, the overhead will be unacceptable.

There are no routing tables, in the sense that there is no one entry for each possible destination, but nodes have to maintain a cache in which are stored all source routes that they are aware of: the cache is continuously update and it is possible have more than one entry for the same destination (helps in case of failures).

Route discovery

When a source has to sent a message to an unknown destination, it sends a Route Request message: this message contains:

. the id of the source;

. the sequence number;

. the destination id;

. the partial route;

. the TTL;

. an id to identify the message;

. the cost (metrics could be: the number of hops, energy consumption, monetary cost).

If after a timeout no replies are received, the source sends a new RREQ and increases the sequence number.

Each intermediate node has to follow this procedure:

. check if the RREQ is recently seen by checking the sequence number: in positive case, it has to discard the request;

. check if its address is present in the partial route; the two checks are needed because cache memories are limited so few RREQ can be stored; it can happen that the particular RREQ has been seen, but not stored : the two checks allow to not process twice the same RREQ; as before, in positive case the RREQ has to be discarded;

. if after the two check procedures the node does not know the destination, it has to append his address to the partial route, store the RREQ in his cache, decrease the TTL and broadcast the packet;

. if after the check procedure the node discovers that itself is the destination, it has to answer with a Route Reply; RREP includes all node crossed by the RREQ, but not necessarily follows the same path: if there are unidirectional links, it is possible use any cache that contains a route to the source and if there are not, it is possible send a RREQ and piggyback the RREP.

. after having received the reply, the source can start send packets (intermediate nodes have not to store the route since the source already does it).

Through route caching is possible avoid rebroadcast RREQ if the node knows already the destination: it replies immediately with the same procedure of the destination mentioned above. This allows:

. to speed up the discovery procedure;

. to reduce overhead.

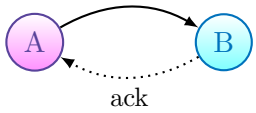
Route maintenance

Along a route, each node is in charge of the correct transmission confirmation: it is realized by acks. They can be:

. realized at MAC or link layer;

. passive acks: station A hears node B sending to C;

. DSR-acks (at network layer): the node that transmits has to require an ack; if the link is symmetric, this procedure is not so much helpful as the asymmetric scenario.



When an akc is not received and the maximum number of ack-req is exceeded, the link to the next hop is assumed to be broken, therefore is marked as invalid, all caches that include that link are removed and an RERR (Route error) is sent to every source that has a route containing that link.

Nodes that hear a REER remove from their caches all routes that include the broken link and the transport layer should retransmit data because packets are assumed to be lost.

Some optimization are available:

. *promiscuous mode*: nodes detect and demodulate also packets not destine to them; this allows to store more new routes, but the drawback is that much more energy is consumed;

. *packet salvaging*: intermediate nodes, when a failure occurs, can store a packet and re-routed it by replacing the old route with the new one; double salvaging is not allowed to not incur in loops.

Pros/Cons

✓ soft state and support of unidirectional links;

✓ source routing:

  . no routing decision taken by intermediate nodes;
  . learning of new routes by forwarding packets;
  . loop free routes (the entire route is already given);

✓ caching:

  . reduce overhead;
  . possibility of having more route for the same destination;

✗ RREQ flooding can be huge:

  . the possibility of collisions between RREQs propagated by neighbors is high;
  . storm problem: contention between RREPs due to local caches;

✗ on demand routing implies delays/jitters;

✗ header may become too large: issue in huge networks;

✗ cache validity affect performances.

AODV

AODV (Ad hoc On demand Distance Vector) builds routes on demand and maintains some of them, to few destinations, in small routing tables. Each node maintains:

. a routing table;

. a broadcast id for RREQs;

. a sequence number for each destination route (DSN).

Routing tables are valid for a given period specified by the field *active route timeout.*

The route discovery procedure is very similar to DSR: the main difference is that RREQs contain a filed called *destination sequence number*; it includes:

. the destination sequence number (DSN) for the *forward route*;

. the destination sequence number (DSN) for the *backward route.*

Indeed, when a path is discovered, two *routes* are established:

. one forward,

. one backward,

therefore *symmetrical* links are required.

When the RREQ is sent, an intermediate node follows a similar procedure of DSR and it is in charge of checking the freshness of routes (the one with higher DSN). RREPs are sent when a node discovers that it is the destination: they include the DSN that has to be increase before being sent. RREPs follows the reverse path (backward) and when the source receives them it can established the forward route.

Data is always sent to the next hop by using routing tables because here, source routing ins not implemented. AODV favours the least congested route since the destination has to reply immediately to the first RREQ that receives.

The maintenance phase is ensured thanks to Hello packets exchange: periodically neighbors send them to maintain local connectivity. If they are absent, the node assume that the given link through which is connected to that node is broken. When a link failure is detected, the intermediate node is entitled to increase the DSN and mark the link as inactive: this helps to recognize fresh information and avoid loops.

Pros/Cons

- ✓ No long headers because source routing approach is not used.
- ✓ Nodes maintains routing tables whose entries describe just current routes in use.
- ✓ DSNs allows to avoid invalid/broken links and prevent loop formation.
- ✗ Unused routes expire even if the topology is not changed.
- ✗ Multiple routes are not supported.
- ✗ Symmetrical links are required.

QoS Routing

Costs that are associated to each link can be:

. transmission power;

. transmitter's residual power;

. traffic load;

. residual bandwidth.

Costs are collected through Routing Requests and it is possible choose the minimum by considering all possible routes to a given destination. Paths are identified by sequence number and costs, therefore the destination answer, to a Routing Request with Routing Reply, only if the route cost is the smallest than all other received.

Optimal decisions have to be constantly updated implying large overhead; moreover the mobility that force the system to be highly dynamic makes hard maintain precise link state information.

Routing in VANETs

Approaches discussed before are not suitable because:

. preventive approach might use stale information due to highly dynamic topology;

. reactive approach imply high latency due to route discovery phase.

The best solution is use a *geographical routing* because nodes are easily identify by their position (through GSM for example). By using Hello messages, nodes can acquire the position of their neighbors; indeed, is assumed that destination coordinates are known and included in data packets:

. easily if they are RSU;

. more difficulty if they are OBU because they can move so their position changes over time.

The forwarding occur in a greedy manner: packets are sent to the neighbor whose position is closest to the destination. This method is susceptible of *dead-end* problem (solvable through *right-hand rule* or putting in hello messages also information on 2-hop neighbors to make better decisions).