Point-to-point routing

Readings:
Point-to-point message routing:
(Optional) Brad Karp's powerpoint slides on multi-hop wireless networks
Johnson, Maltz: Dynamic Source Routing (DSR)
Hu, Johnson: Caching strategies for on-demand routing protocols

Next time, we'll continue with routing:
Perkins, Royer: Ad hoc on-demand distance-vector routing (AODV)
Chen, Murphy: Enabling disconnected transitive communication in mobile ad hoc networks
Link-reversal algorithms:
Gafni, Bertsekas: Distributed algorithms for generating loop-free routes in networks with frequently changing topology
Park, Corson: A highly adaptive distributed routing algorithm for mobile ad hoc networks (TORA)
Busch, Surapaneni, Tirthapura: Analysis of link reversal routing algorithms for mobile ad hoc networks

And then we will move on to the next batch of papers on routing:
Rao et al. Geographical routing without location info
Fang et al. GLIDER: Gradient Landmark-based distributed routing
Fonseca et al. Beacon vector routing

1 Introduction

We're dealing with a new problem:
-Routing a message from a particular source to a named destination, in a mobile ad hoc network.

This is similar to the basic IP routing service provided by the Internet—send a message to a particular IP address. But the problem is much easier in the Internet setting: there, we have fixed infrastructure of wires, routers, backbone networks, etc. to rely upon.

The main problem in our setting is that we don't initially know where the node is. However, this is still an important problem, in the wireless setting. It has been widely studied, and proposals are currently being put forth for practical routing methods. The initial proposals are/were based on approaches used in the Internet, not taking special advantage of the nature of the wireless setting. We will begin today with some of these: Dynamic Source Routing (DSR), Ad-Hoc On-Demand Distance Vector Routing (AODV), and (briefly) a little paper proposing a variation that tries to *take advantage* of mobility.

Next time, we'll focus on a particular strategy that traces back to the 80s, work by Gafni and Bertsekas: link-reversal algorithms. We'll see how this idea has evolved into algorithms like TORA that are proposed for wireless networks. Then we'll move on to what is sometimes called "location-free routing", by which they mean routing without explicit use of geographical information. These

methods use some kind of substitute for geography. Finally, we move to approaches that actually use real location information: Geocasting, location-aided routing (LAR), compass routing, etc.

Somewhere in here, we will also consider a related problem: Keeping track of the geographical locations of particular nodes ("location services"). Now it's scheduled to come before the geographical routing material, but we might move it later.

2 Overview

This is from Brad Karp's slides on "Multi-Hop Wireless Networks". He describes the problem, the setting, and some solutions. He also discusses capacity, which we won't talk about.

2.1 The problem and setting

-Wireless networks.
-Ad hoc, no fixed structure (no hierarchical structure as in the Internet).
-Dynamic; stationary and/or mobile nodes; may change rapidly.
-Scale: May be small or large, even Internet-scale.

For some examples, consider: rooftop networks (customer-owned and operated radios, stationary, large-scale), or floating sensors (mobile, large-scale). Other settings could also include: Ad hoc meetings of friends in unusual places, rescue workers in disaster areas, or soldiers in battlefield situations.

The problem is that of multi-hop message routing: a sender anywhere in the network wants to send a message to a particular named node somewhere else in the network, but doesn't know initially where the destination node is.
As a subproblem, we want to set up a route (an end-to-end path) to that node. Routes may need to change, in response to network changes.

We'll see (later, Chen-Murphy) that, some strategies don't actually set up complete routes, but just send the messages one step at a time. Hence, it's better to think of the real problem as the higher-level task of sending the messages, rather than the lower-level task of setting up the routes.

Here are some of our goals:
-Accommodate large scale, mobility.
-Minimize amount of communication overhead for sending data messages.
-Maximize packet delivery success rate.
-Minimize latency of packet delivery.
-Minimize route length (try to approach the best-possible number of hops).
-Minimize amount of state that must be maintained at each node.

Some strategies for coping with scale are:
-Hierarchy: This is good when level boundaries are relatively fixed, and can be chosen by administrative authority.
-Caching (e.g., for source routes).
However, these don't work so well for dynamic or unstructured networks.

2.2 Routing algorithms for traditional wired networks

We have two main methods: *Link-State Routing* and *Distance-Vector Routing*. Both methods try to set up paths with fewest hops ("shortest paths"). Both require a lot of state ($O(n)$) per router.

Link-State Routing:
Each router bcasts to all other routers in the network its view of the status of each of its adjacent network links. Each router assembles a complete picture of the network, and uses it to determine shortest paths, which tells it which routes to try when sending messages.

Distance-Vector Routing (Bellman-Ford-style):
Each router keeps track, for each possible destination, of the shortest distance (number of hops) that it knows to that destination, and the first hop along a minimum-hop path. Each router bcasts to its neighboring routers its view of the distances to all hosts. Each router adjusts its minimum distance estimates based on information from its neighbors. This must refresh periodically: you ignore your own estimate and recalculate an estimate based on neighbors' estimates.

Both algorithms require that, in response to link changes, information must be propagated to all routers. This is not so good in the wireless setting; we don't have routes, and there are potentially many links between nodes.

2.3 Algorithms for wireless ad hoc networks

For ad hoc routing there have been many algorithm proposals: DSDV, DSR, AODV, TORA, GPSR, ZRP,..., and many approaches: distance vector, source routing, geographic, proactive, on-demand.

Destination-Sequenced Distance-Vector Routing (DSDV):
This was an early proposal for routing in wireless networks, by Perkins and Bhagwat. It's a version of a Distance-Vector algorithm, where each mobile node is treated as a router. Each destination d initiates a Bellman-Ford-like DV algorithm, sending its best distance information out, and propagates as usual, each node keeping track of the shortest-hop path it has seen.
But now, to cope more systematically with changes, d uses *sequence numbers*. These allow other nodes to determine which information is newest, by comparing the sequence numbers. Other nodes select routes based on newest information as the dominant factor, then fewer hops as the secondary factor.

DSDV doesn't scale well: It depends on periodic advertisement and global dissemination of connectivity information. It also has a control message overhead of $O(n^2)$, just to set up route information to all destinations once (essentially, a shortest-paths tree to each destination node). This route information will have to change as the network changes. Finally, it requires each node to maintain a complete set of route entries, one for each destination.

The approach taken to try to reduce this communication overhead and storage usage is to try to create routes on-demand. We'll see the most important examples of this approach today.

3 Dynamic Source Routing (DSR)

3.1 Introduction

They start with the observation that conventional wired-network routing protocols don't work well in mobile ad hoc networks: they require periodic routing advertisements, regardless of need. Instead, DSR uses *on-demand* routing: it only generates routes (and only incurs costs of routing traffic) when they are needed by the message forwarding protocol.

Basically, when a node has a message to send, it tries to find a route to the destination. It performs *route discovery*, by flooding a query packet to try to find a good route to the destination. It learns the entire route. It attaches the entire route to each data packet it sends (*source routing*). It caches the routes so it doesn't have to search for new routes for each message (or each packet). If the network is large and or/changes rapidly, the routes tend to break frequently, which makes caching less effective. Hence, there is some tradeoff in how long to keep routes while having them be valid.

Advantages of DSR:
They claim good performance under a variety of conditions, with generally low communication overhead, high reliability and low latency of message delivery, and nearly-optimal routes:
1. It sends information only when needed. Specifically, DSR produces and maintains only needed paths; there are no background periodic routing advertisements. Ad hoc networks typically have many links, compared to wired networks; LS and DV would send info about all these links, which is just too many. Use of DSR leads to savings in communication overhead, battery power.
2. There is little overhead during stable periods. DSR just keeps reusing already-determined routes. This leads to low-latency, low-communication reliable message delivery, in stable situations.
3. It adapts reasonably quickly to changes in the network—diagnoses broken routes and finds new ones. LS and DV aren't designed to handle continuing/frequent topology changes—just occasional router failures and recoveries, or changes in congestion levels. In the mobile setting, DV may take a long time to converge to new stable routes.
4. DSR works even without bidirectional communication (though it can take advantage of bidirectional links when available). DV and LS are generally designed for networks with bidirectional links; if some links aren't bidirectional, they may produce unusable paths.

3.2 Assumptions

We assume that nodes are willing to "participate fully" (forward traffic on behalf of others, etc.). Their examples use fairly small network diameters. (This suggests that DSR may not scale so well.)
Mobility:
-Hosts may move at any time.
-Movement speed is much lower than time for packet transmission and local processing. (This seems reasonable.)
-Hosts don't move so fast as to make route computation useless.

Hosts have a "promiscuous receive mode", that allows them to receive every packet they hear, not filtering by destination address. (They have optimizations taking advantage of this.)

3.3 Basic Operation

3.3.1 Overview

A sender puts the entire route in the packet header, forwards the packet to the first host in the route, and successive hosts forward the packet (source routing).

Each host maintains a route cache, caching some source routes it has learned, each with an expiration time. When a sender wants to send a packet to a destination d, it first checks its route cache for a source route to d. If it finds one, it uses it. If not, it uses the *route discovery* protocol to try to find one. It processes all other packets in the meantime.

For route maintenance, the source monitors correct operation of its cached routes; if one isn't working, it removes it from the cache. If a route currently being used is discovered to be non-working, then the source may do route discovery again, and resend on the new route.

3.3.2 Route discovery

Any host s can discover a route to any other host d in the ad hoc network. Route discovery works by simply flooding a packet, looking for a path that works. Initiator s bcasts a "route request" (RREQ) packet, containing the id of the source and destination. The result should be a "route reply" (RREP) containing a full path.

A RREQ packet contains:
-source s, target d,
-request id, a sequence number to keep track of separate route requests, which allows pruning out of duplicates, and a
-route record, which accumulates a record of the sequence of hops (path) the packet takes during discovery.

Processing a route request packet:
A host that receives a route request packet discards it if is a duplicate, according to request id. It also does loop detection: if the host's address already appears in the route, this is a loop, so the host discards the packet. If the host is the destination d, the route has been found, and the host sends RREP with the final route back to s. Otherwise, it appends its own address in the packet and re-broadcasts.

How can you send the RREP back to s?
-You can reverse the route in the packet, but this requires bidirectional communication.
-For the case where there is no bidirectional communication, they do something cuter; they use essentially the normal message-forwarding protocol to send the RREP: Use a route to s that is currently in your cache, if any. Otherwise, initiate another route discovery (flooding), with a new RREQ message, but this time piggyback the route information on the new RREQ message.
Q: Why isn't this circular?
A: We're not counting on this second route discovery to succeed in providing d with a route from d to s—rather, we're just using the flooding capabilities to get the info about the route from s to d back to the source s. But in fact, when s receives this new RREQ, it puts the piggybacked path into its cache, and now it does have a path from s to d. So it could now reply normally to this new

RREQ, sending along the new path from s to d, the info it has just learned about the path from d to s.
On the other hand, this would not work without piggybacking:
If we allowed d to just do route discovery to s without piggybacking the path from s to d on the new RREQ message, then s would not have a path to use to return the RREP to d (so it would have to call route discovery again...which is circular).

3.3.3 Route maintenance

Conventional routing protocols send routing updates continually, in the background; this helps them to maintain their routes—ensuring that they remain usable and optimal. But DSR doesn't use such background messages. Instead, while a route is actually being used to send a message, a separate *route maintenance* protocol monitors the operation of the route and informs the sender of any problems.

How it does the monitoring:
1. Suppose that the wireless network provides data-link-level acks, and reports the non-occurrence of such an ack to higher layers. Then the route maintenance protocol can use this negative-ack mechanism to diagnose the failure of individual hops along the path in the header of a message. A host discovering such a local failure then sends a "route error" message to the sender of the current message, indicating the particular hop that failed. When the source s receives this, it truncates all the paths in its cache at the failed hop.

2. If the wireless network doesn't provide data-link-level acks, then a similar diagnosis can be done with other kinds of single-hop acks, such as:
-"Passive acks" (a node tries to overhear the next transmission, by the next host along the path).
-Some application-level ack info, if available.
-Adding acks explicitly to the message-forwarding protocol, e.g., by including a bit in packet header for a packet being forwarded, requesting an explicit ack.
In all cases, the diagnosis is reported to the route maintenance protocol, which can handle it as in case 1.

Q: How do we get the route error message back to the sender s?
A: -The diagnosing host may have a route to s in its cache—if so, it uses it.
-It could reverse the route from the packet (if communication is bidirectional).
-It could piggyback on a route discovery, as for the RREP message discussed above.
-It could perform an entire route discovery from the diagnosing host to s, before sending the route error message (without doing piggybacking).
–Q: But why is this not circular?
–A: It was when we hit a similar situation for RREP messages, but route-error is a bit different: this is likely to yield a path with fewer hops than the original path that has been broken, which is a smaller subproblem—this decrease might break the circularity.

We could also do route maintenance end-to-end by using an explicit ack for the data packet, from the receiving destination d to the source s. This type of diagnosis gives "coarser" information—a source could only delete one route, not truncate many routes as in the case of single-hop diagnosis.

3.4 Optimizations

They continue by describing several optimizations.

3.4.1 Full use of the route cache

1. How should a host store the routes in its route cache?
It could keep a list of routes, indexed by destination. It seems there is just one per destination in this paper, but in the follow-on paper by Hu and Johnson, they seem to allow multiple routes to the same destination. Assuming that we have only one, it is likely that some routes are prefixes of others; in fact, it may be possible to organize the route cache as a tree of routes, from this host to other hosts.
A tree representation could be much more compact than a list. It could also facilitate certain optimizations. For example, discovering a new, short route to one destination c may allow the host to automatically shorten all the paths that contain c as an intermediate host.

2. When can/should a host add a route to its cache?
-As a result of a successful normal route discovery, of course.
-When it forwards a packet along a designated route, it could copy the remainder of the source route, from itself to the destination, into its own cache. (Should it? is this likely to be useful?)
-When it forwards a RREP message containing a particular route, it could add that route to its cache.
-More generally, it could add any path it overhears information about.

3. A host can use its own route cache to avoid propagating a RREQ packet:
If the host already has a path to the intended target d of the RREQ (which would not introduce a loop), it can append it to the path from s to itself that already appears in the RREQ, to obtain a candidate path from s to d. It then send this in a special RREP message back to s. (This sounds like an important optimization.)
There is a problem with this: Multiple nodes might reply, causing packet collisions and returning different paths. They have some ad hoc solutions for this, e.g., introducing random delays to avoid collisions, or listening for indications that others have already replied with a shorter route.

4. Getting path info from neighbors' caches:
Hosts could query immediate neighbors for paths to the destination and use those. It could then do 2-hop neighbors, etc., in an "expanding ring" of searches. We allow the source to set a limit on desired path length, to limit the search to reasonable-length paths, and limit the number of hops along which the RREQ packet is propagated. They use a limit of 10, which suggests that the scale of the networks they are interested in is 10-hop.

3.4.2 Other stuff

1. Piggybacking data on route discoveries:
A node does route discovery because it wants to send some data. It could piggyback (some) data on the route-request messages. Then, if the search gets completed by some node from its own cache, the node has to assemble a new packet containing the data and send it to the destination.

2. When a shorter route becomes possible, how do we get it to replace an older, longer route?

If an older route becomes invalid, or times out, it will disappear and a new route discovery should find the shorter route. We could speed this process up: if a host learns somehow about a short route between two other nodes s and d, it could send an unsolicited RREP to s to help it out.

3. Improved handling of errors (lots of little things):
-To avoid generating too many RREP messages in the case of a partition, we use some kind of backoff scheme to limit the rate of retries.
-Overhearing route-error packets can allow hosts to remove/truncate paths from their own caches.
-Remembering negative information: When a node learns that a hop isn't working, it should, for a while, not accept path suggestions from its neighbors that include that hop. However, it should not do this for too long—what if the hop starts working again?

3.5 Performance evaluation

They did a (packet-level) simulation. They varied parameters in the protocol, number of nodes, pattern/speed of movement, and distribution of hosts in space. Basically, the simulated mobile hosts in a medium-sized room. The hosts were moving at walking speeds, and had a 3-meter transmission/reception range. (They claim the results are also valid for vehicles, which go at faster speeds, over a larger region, because (they claim) the parameters scale appropriately (?).) They looked at random initial placements, random pause, random new locations, and random speed.

Results:
-For all but the shortest pause times, the total number of packets transmitted is very close to optimal (for communicating actual data).
-Nearly all data packets are successfully delivered, because route maintenance works well enough to detect when a route is no longer working. They claim DV approaches would be unable to keep up with changes and would be unable to deliver many data packets.
-The protocol finds and uses close to optimal routes.

3.6 Discussion

DSR is similar to some wired source routing protocols. However, in wired networks, orderly flooding is ensured by the lower-level network. Here, the network doesn't provide such help, so the algorithm has to do some work to make flooding orderly and efficient, such as:
-Caching of recent RREQ packets to prune out repeats.
-Overhearing and caching routes to stop flooding RREQs.

Movement speed: If nodes move very fast, the best strategy must be just flooding. If nodes were slow, little flooding should be needed—stable paths could be maintained, requiring little overhead for data transmission. DSR moves easily between these two extremes (depending on the rate at which RREQs are initiated).

How should this work in a setting that combines ad hoc networks with a wide-area network (Internet)? The WAN should be reachable by some, but not all of the wireless nodes—so we would still have to use ad hoc network protocols locally.

4 Caching policies for DSR

This from the paper by Hu, Johnson. They consider DSR and similar protocols, in which a source attempts to discover routes to a destination only when it actually has data to send. Caching is used to avoid too many route discoveries.
This paper simply considers different design choices for such caching strategies, focusing on DSR, for: Cache structure, Cache capacity, and Cache timeout settings. They evaluate the effect on several measures of routing success:
-Ability of a protocol to successfully route packets to their destinations (what percentage).
-Latency of delivering data packets.
-Amount of communication overhead generated by the routing protocol.
-Optimality of the routes relative to shortest paths.

The results apply directly to DSR, but they claim they should be significant for other on-demand protocols as well.

They do their study using simulations (of small, 50-node networks). They modelled MAC control, contention, collisions, wireless signal strength and propagation delay, carrier sense,..., all using ns-2. They had 50 different movement scenarios, from 5 mobility models.
Also, they define some "mobility metrics" (such as Johansson's geometric metric and some new ones) and study how well they correlate with performance.

4.1 Overview of DSR

We just covered this. Key aspects used here are:

Route discovery: This is used only when s wants to send a packet to d and doesn't already have a route to d. A RREQ packet terminates when it either reaches d, or a node that already has in its cache a route to d. A RREP packet returns a route, which is cached by s. Node s may receive, and cache, multiple routes to the same destination.

Route maintenance: This is used only when the route is actually being used to send a packet. It's performed at each hop as a data packet if forwarded along a source-specified route. A diagnosing node sends notification of an error to the original sender. The sender would then remove the broken link from its cache. The diagnosing node may try to salvage the packet, continuing it to its destination on an alternative path.

Nodes may learn helpful routing information by eavesdropping on packets (this is promiscuous mode).

4.2 Caching strategy design choices

4.2.1 Cache structure

There are two kinds of cache structures we'll consider, path and link.
Path cache (caches complete paths):
-It's simple to implement.

-It guarantees routes are loop-free.
-It's simple to search for a route to a given destination.
-However, it is somewhat "coarse-grained". It doesn't make it easy to ues all available information about the network links, and can't piece together parts of paths it learns about to form other paths.

Link cache (just cache individual links):
-It's easy to store stuff in it, but hard to find a route—this involves a graph search.
-However, it should be possible to pre-calculate some routes and thus remove them from the critical path for latency of data-sending.

They don't talk about tree caches, as described in the DSR paper. This is probably because they are allowing caching of multiple paths to the same destination—not something that can be maintained in the form of a tree.

4.2.2 Cache capacity

Link cache:
-We can allow enough space to keep track of every link, network-wide.
-It's not prohibitive.

Path cache:
-Too much space would be required to store all paths, in a naive data structure. So they consider effects of different limits on number of paths cached.
They thought a larger cache would be better, but found out this isn't true.
Q: Why not?
A: Bigger cache means more stale paths.

They also considered dividing the cache into two halves, one for paths that have actually been used to send a message, and one for the others that haven't. This division is supposed to help recently-used paths to remain—implementing a sort of LRU policy. This is called "generational cache".

4.2.3 Cache Timeout

They considered this only for link caches, since for path caches, they just relied on the capacity limit to remove old paths. They considered:
-Static timeout: Each link is removed after a specified (fixed) time. (They observe something similar to the cache capacity situation, where a large timeout can be bad because it allows more stale data.)
-Adaptive timeout: A node guesses a good timeout based on properties observed for the link. Also, it may increase the timeout value for a link when it is used.

4.3 Caching algorithms studied

There were just too many. They studied:
Path caches:
-No capacity limit.

-64-path capacity limit, eject LRU path.
-Generational path caches.

Link caches:
-No timeout, no expiration.
-Links expire after 5 seconds (of being unused?).
-Links expire according to a stability table (adaptive).

They sometimes impose a constraint on choosing routes: Consider the routes that have the latest timeout (minimum over all their links), and choose the route among these having the fewest hops. They also consider the ideal link cache: one in which each link is removed from the cache exactly when it actually fails.

4.4 Methodology

They used ns-2, which they extended to support wireless and mobile networks. They model signal strength, RF propagation, propagation delay, contention, capture, interference, and mobility.

The DSR performance metrics they were concerned with were:
-Packet delivery ratio.
-Packet delivery latency.
-Communication overhead.
-Path optimality.

4.5 Mobility models studied

This seems to be an effort to define some measure of amount of mobility, and try to correlate it with performance of DSR. I don't think this work has led anywhere.

Mobility model specifications included:
-Brownian motion.
-Column motion.
-Random Gauss-Markov motion.
-Random waypoint: this has been used quite a bit. You choose a random point to go toward, choose a random speed in some interval, choose a random time to wait there, repeat,...
-Pursue motion.

4.6 Simulation results

4.6.1 Effects of Cache Structure

They found that link caches out-perform path caches for packet delivery ratios and for routing overhead.

Packet latency is primarily affected by:

-Time spent by a packet waiting for Route Discovery to complete before a packet can be sent, and
-Time spent in Route Maintenance detecting broken links and salvaging the packet by sending it on alternative route.

For some motion patterns, link caches are better; for others, path caches were.
All the caching algorithms achieve good path optimality. The maximum-lifetime criterion for choosing paths performs well.

4.6.2 Effects of Cache Capacity

Using a large path cache turned out to be bad, because it meant that the cache contained many stale paths. Basically, we want cache entries to be removed in a way that adapts to the network changes, not because of arbitrary capacity limitations.

4.6.3 Effects of Cache Timeouts

Effects are similar to those of limited capacity: No timeout, or large timeout, is bad if it keeps around stale info. The bad performance is, not surprisingly, most evident in the scenarios in which the network is most dynamic.

Routing overhead is affected by choice of timeout much more than the packet delivery ratio. That seems to be because, when links are timed out for no good reason, they can be rediscovered, and will still work—but it costs overhead to do it.

They say that the timeouts should be chosen to relate well to various network parameters and parameters of the routing protocol. Their experiments indicate this, for static timeout values. However, they don't say what the correlation should be—they just give the best values for some particular parameter values they tried.

4.7 Conclusions

The caching strategy is key to performance of on-demand routing algorithms for wireless networks, including DSR, TORA, AODV, etc. Caching is important for avoiding the overhead of route discovery, but it carries the risk of retaining routing info after it's invalid.

Main findings:
-The performance of adaptive caches is comparable to that of well-tuned static caches.
-Link caches are better than path caches.
-They identified some subtle relationships between cache timeout policies and cache capacity limits, and observe some specific effects of timeout policy and capacity on the performance metrics (such as packet delivery ratio, overhead).
–Path caches with unlimited capacity perform worse than those with reasonable capacity.
–Similarly, link caches with no timeout aren't good.