Wireless Ad-Hoc Networks: Routing and Forwarding Techniques

Pawel Gburzynski

Department of Computing Science
University of Alberta
&
Olsonet Communications
No wires, no infrastructure, mobility, low cost

The wireless medium is broadcast in a nasty way:
What is there to accomplish?

The optimization criteria:
- number of hops (delay)
- spatial reuse (throughput)
- quality of service (loss, delay, jitter)
- power control (durability)

The problem: **routing**, mostly how to emulate wires?
- given a source and destination, find the best path
- then forward the packets along that path
- oh, and be prepared that some of those links may disappear on you as you go along
Two schools

Proactive: \((\text{DSDV, CGSR, OLSR})\):
- perpetual background activities aimed at detecting and maintaining routes before they are needed

Reactive: \((\text{AODV, DSR, TORA})\):
- detect routes as they become needed; maintain them while they are being used
Two types of traffic (regardless of the school)

**Data traffic:** the packets traveling from sources to destinations, as viewed by the network's application.

**Bureaucracy:** the extra packets that are needed to establish/maintain connectivity.
Medium Access Control (802.11)

Three objectives:
- to resolve contention among multiple nodes trying to transmit at (about) the same time
- to do something about the notorious hidden terminal problem
- to provide for data-link reliability

<table>
<thead>
<tr>
<th>RTS</th>
<th>CTS</th>
<th>DATA</th>
<th>ACK</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RTS</th>
<th>DATA</th>
<th>ACK</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RTS</th>
<th>CTS</th>
</tr>
</thead>
</table>

SIFS | DIFS | NAV | contention |

random back-off at the perceived end of activity
The common “proactive” component

Nodes keep themselves aware of their neighborhood with the help of HELLO messages. Those messages are broadcast (locally) every now and then.

If you do not receive a HELLO message from one of your past neighbor for a while, you assume that that neighbor is no more.
DSDV

The generic formula for Distance Vector Routing:

\[ D_X[Y,Z] = c(X,Z) + \min \{D_Z(Y,w)\} \]

- cost from node \( X \) to \( Y \) via neighbor \( Z \)
- cost from \( X \) to \( Z \)
- the lowest cost from \( Z \) to \( Y \) over all neighbors of \( Z \) (\( w \))
- minimum hop count?
In a nutshell

Nodes maintain and advertise their cost (distance) tables. Such a table (at node A) gives the minimum cost (e.g., hop count) of reaching every other node from A.

Based on the advertisements received from its neighbors, a node can update its own routing tables (see the formula) and send out new advertisements (which can be incremental).

This operation is distributed, which implies coherence problems: loops, slow convergence after “bad news”.
The objectives are to:

- Avoid loops
- Provide for prompt propagation of fresh updates
- Efficiently communicate incremental and/or important changes
- Avoid the caveats of over-eager update propagation (wait until the route has become “stable enough”)
# Routing table entry

<table>
<thead>
<tr>
<th><strong>Destination</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong> (typically the number of hops)</td>
</tr>
<tr>
<td><strong>Serial number</strong> (usually inserted by the destination)</td>
</tr>
<tr>
<td><strong>Timer</strong> (for expiration)</td>
</tr>
</tbody>
</table>

Note that the entry has its ultimate origin in the destination, which must have reported itself at some point with the cost of 0.
Example

D  N  H  SN
1  2  2  406_1
2  2  1  128_2
3  2  2  231_3
4  4  0  710_4
5  6  2  620_5
6  6  1  114_6
7  6  2  982_7
8  6  3  426_8
Example

<table>
<thead>
<tr>
<th>D</th>
<th>N</th>
<th>H</th>
<th>SN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>?</td>
<td>∞</td>
<td>407_2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>128_2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>231_3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>710_4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
<td>620_5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>114_6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>2</td>
<td>982_7</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>3</td>
<td>426_8</td>
</tr>
</tbody>
</table>
Example

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>408_1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>128_2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>231_3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>710_4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
<td>620_5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>114_6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>2</td>
<td>982_7</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>3</td>
<td>426_8</td>
</tr>
</tbody>
</table>
```
An entry with a higher serial number supersedes an entry with a lower number.

An entry with the same serial number wins if the number of hops (the cost) is smaller.

Being jumpy with propagating information that will be shortly overridden may result in a significant overhead.

\[ t_{a}^{(n+1)} = \alpha t_{f}^{(n)} + (1 - \alpha) t_{a}^{(n)} \]

average “time until stable”
CGRS: a scalable descendant of DSDV

Reducing the complexity of maintaining the routing tables via a hierarchy.
OLSR

- the MPR set of A
- 1-hop neighbor of A
- 2-hop neighbor of A

MPR
MPRs are responsible for connectivity

\[ A \in \text{MPR}(S) \quad B \in \text{MPR}(A) \quad D \in \text{MPR}(F) \]

The complexity of routing is reduced because:

- for every \(<S, D>\) pair, the route involves only the usually quite narrow subset of the potential nodes-relays, namely a chain of MPRs
- disseminating route information is more efficient because of simplified broadcasting
AODV

A node S trying to establish a session with node D initiates a path discovery operation.

RREQ messages probe the network forward looking for the destination.

RREP messages go backward setting up the routing tables at the intermediate nodes.

The scheme relies on caching. Routing table entries expire and are discarded when routes are not used.

RERR messages (traveling backward) convey information about broken links.
Path discovery

**backward**

- **D**
- **S**
- **broadcast**

**forward**

- **D**
- **S**
- **unicast**
- **routing table entry**
- **timeout**
Some quirks

<table>
<thead>
<tr>
<th>RREQ</th>
<th>D</th>
<th>S</th>
<th>ID</th>
<th>SSN</th>
<th>DSN</th>
<th>HCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RREP</td>
<td>D</td>
<td>S</td>
<td>ID</td>
<td>DSN</td>
<td>HCN</td>
<td>LTM</td>
</tr>
</tbody>
</table>
Routing table entry

<table>
<thead>
<tr>
<th>waiting for RREP</th>
<th>RREP received</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACK</td>
<td>BACK</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>Timer</td>
<td>DSN</td>
</tr>
<tr>
<td>SSN</td>
<td>Timer</td>
</tr>
<tr>
<td></td>
<td>HCN</td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
</tr>
</tbody>
</table>

\[ DSN < DSN_{\text{RREP}} \quad \text{or} \quad DSN = DSN_{\text{RREP}} \quad \text{and} \quad HCN < HCN_{\text{RREP}} \]
Which approach (proactive vs. reactive) is better?

Performance criteria depend on the application:
- packet delivery fraction
- maximum throughput
- stability of delays

Observed performance depends on network properties:
- topology, density
- quality of links (the propagation environment)
- mobility patterns

The protocols have many adjustable parameters. How to set them right?
One factor, often ignored by academics, is:

The “true” cost, which determines the suitability of a given solution for a particular commercial project.

Solutions that poorly scale to the network size may require a lot of memory and/or processing power.

Trade-offs are often desirable, e.g., trading the route quality for the device footprint. This allows the commercial vendor to offer a wider range of products (with different price tags).

Also note that in most “actual” deployments of ad-hoc networks, the (application) packets are short.
In a typical, random, planar, balanced network with $N$ nodes:

- $M$ (the number of parallel sessions) $\sim N$
- $R$ (the number of nodes occupied by a single session) $\sim N^{1/2}$
- $S$ (the number of session passing through a single node) $\sim N^{1/2}$

AODV needs $\sim S$ memory per node while DSDV needs $\sim N$ memory per node; thus AODV (and generally reactive protocols) better scale to the network size.

On the other hand, one would expect DSDV to be more responsive to new sessions.
Responsiveness to mobility

PDF

Pause time

AODV
DSDV

low mobility
high mobility

0.7
0.8
0.9
1.0

100 300 500 700 900
Session start time

Network size

AODV
DSDV

msec

1.0
3.0
5.0
7.0
9.0

10 50 90 130 170

Network size
DSR: Dynamic Source Routing

The source receives back the full path, possibly a few alternatives to choose from.
More on DSR

Nodes are allowed to overhear **RREPs**, as well as **DATA** packets, and cache the overheard routing information.

\[ s, v_1, v_2, v_3, \ldots, v_k, d \]

Nodes can issue **RREPs** based on cached routes. They may not be optimal, but the same mechanism as in AODV will kick in.

**RERR** is sent as before invalidating respective cached entries on its way back. The source may decide to use an alternative cached route, if it is available.
Hop limits to scope the propagation of RREQs. Why bother discovering very long (and thus volatile) paths.

An RREQ with the hop limit of 0 will poll the neighbors for their routes. Note that HELLO can be combined (or equivalenced) with such an RREQ.

Proactive shortcuts: nodes transmitting unsolicited RREPs.

Sources may repeat RERRs they receive to flush their neighbors' stale cache entries.
TORA (link reversals in DAGs)

(a) original state

D

R

(b) 1st iteration

R

the only outgoing link

no reverse here because there exist paths to D

(c) 2nd iteration

R

D

R

(d) 3rd iteration

R

D

(f) final state
Shortcomings

While link reversal may provide many alternative paths to the destination, none of them need to be optimal.

Will loop if the network becomes partitioned.
TORA

On-demand DAG establishment for a particular destination (via controlled flooding).

The DAG is created using the notion of distance (height), as to reflect the hop metric.

Time-stamping of the link events (disappearance of the last downstream link).

The reversal scheme ensures that the most recent event wins with older events; thus, in the eventual steady state, the routes are consistent with the most recent information.
TORA gives lower re-routing overhead

![Graph showing PDF over pause time for AODV, DSDV, and TORA]

- **AODV**
- **DSDV**
- **TORA**

Pause time

100 300 500 700 900

PDF

0.9 1.0
Specialized routing (stronger prereqs):

Nodes know their geographical locations, e.g., via GPS. People generally tend to abuse this feature.

Planar (compass) routing: push the packet in the direction of the destination. Not as simple as it sounds.

Directional antennas: if we know the direction of the next hop neighbor, perhaps we should narrow down the area affected by our transmission.
A way out
The third school: flooding

Well ... this is the simplest way to make sure that a message gets from point A to point B (minimalistic in terms of the requisite knowledge).

All the sophisticated protocols occasionally resort to some forms of flooding to acquire their wisdom.

All this point-to-point forwarding does seem to miss something. The medium does not really consist of a bunch of wires.
TARP: Tiny Ad-hoc Routing Protocol

No packets other than the transport (application) packets. No neighborhood discovery. No explicit route discovery. **No data-link encapsulation.**

Cache-based. The cache size is flexible and automatically trades off route quality for footprint.

Rule-based. Easy to extend and parameterize.

Aimed at embedded systems and the lowest end of the hardware spectrum, say 1K of RAM.
Note: the 4-way MAC handshake ...

... only makes (some) sense if **DATA** is considerably longer than **RTS/CTS**. Also ...

... it may inhibit legit transmissions, for example: **G** cannot transmit to **H**, although it would be fine ...

... may cause possibly cascading (false) blocking: suppose **E** sends **RTS** to **F**. Note that **E** could in principle talk to **F**, but **F** cannot respond with **CTS**.
Controlled flooding

The rules are based on cached information. If no information is available in the cache, the rule fails.

- am I the recipient?
  - NO: too many hops?
    - NO: rule 1
      - drop
    - YES: drop
  - YES: receive and forget

- too many hops?
  - NO: rule 2
    - drop
  - YES: drop

- rule n
  - fail
  - rebroadcast
The simplest rule: DD

packet signature in cache?

YES

entry expired?

NO

YES

fail

drop

One can experiment with the replacement policy, e.g., make the expiration time proportional to the expected distance from the destination.
## Info stored in packet header

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>destination</td>
</tr>
<tr>
<td>S</td>
<td>source</td>
</tr>
<tr>
<td>s</td>
<td>session tag</td>
</tr>
<tr>
<td>n</td>
<td>packet number</td>
</tr>
<tr>
<td>F</td>
<td>optimal path flag</td>
</tr>
<tr>
<td>L</td>
<td>hop number limit</td>
</tr>
<tr>
<td>h</td>
<td>hops so far</td>
</tr>
<tr>
<td>H</td>
<td>target hop count</td>
</tr>
<tr>
<td>m</td>
<td>focus</td>
</tr>
</tbody>
</table>

**Signature for detecting duplicates**

The **actual** hop count of the last (non-duplicate) packet that has reached the sender.
The SPD rule

cached \rightarrow [S, D, H_{SK}, H_{DK}]

Also, fail every once in a while, with frequency inversely proportional to $H - (H_{SK} + H_{DK})$. 
Path convergence
If \( A \) thinks its packet is on the optimal path, it sets the \( F \) bit in the header.

If \( B \) (or \( C \)) overhears a packet with \( F \) set, whose signature matches the signature of one of its own packets awaiting transmission, and the hop count of \( B \)'s packet is \( \geq \) the hop count of the overheard packet, \( B \) will drop its packet.
It will not help here:

Oh, what the heck! Fault tolerance is getting back in fashion these days 😐.
Fuzzy ACKs

How can you be reasonably confident that your forwarding duties have been fulfilled?

Listen for a retransmission of your packet?

Only sent if the node is not going to drop the packet.

Layering gets in the way
Application-triggered recovery

End-to-end ACKs are generally a better idea in this type of networks than “accurate” point-to-point ACKs.

Setting $m$ in the header of an outgoing packet will instruct nodes to relax their rules. This can be binary (e.g., revert to flooding) or gradual (e.g., increase $e$ in SPD).

$$H + e < H_{SK} + H_{DK}?$$
Other benefits

No data-link layer encapsulation: packets are never addressed to specific next-hop nodes (only to the actual destinations).

Intermediate nodes, i.e., ones that never originate or absorb sessions, need no addresses!
Summary

Lots of protocols, lots of parameters.

Few useful applications of the complex protocols.

The industry demands some ad-hoc networking; they are confused, but they want something different from what we are trying to offer.

There is no end to the demand for trivially simple and extremely cheap devices.

There are alternatives to emulating wires.