Babel
A routing protocol for sparse networks

Juliusz Chroboczek

Laboratoire PPS
Université de Paris 7
jch@pps.jussieu.fr

25 July 2008
Goal: wireless mesh networks

Our goal is to design dynamic routing protocols for wireless mesh networks.

- no distinction between host/router or STA/AP;
- frequent mobility even when nobody moves (variable radio propagation conditions).

The first widely deployed such protocol was OLSR.
Usual routing protocols are designed for *wired broadcast networks*:

- communication is *symmetric*
- communication is *transitive*
- communication is *bimodal*
Radio is not Ethernet

- Communication is not transitive

- Communication is not symmetric

- Communication is not bimodal (marginal links).
Dense networks

Most often, mesh networks are used to replace enterprise Ethernets. These are dense networks:

- a node usually has many neighbours: it is important to reduce redundancy;
- the network remains connected without marginal links: these can be ignored.

OLSR is optimised for dense networks.
Sparse networks

Some networks are sparse (≠ dense):

- urban community networks;
- rural networks.

In such networks:

- reasonable number of nodes (10 to 1000);
- marginal links (> 50% packet loss) are productive;
- long paths (7–8 hops) happen.

Think of them as long, thin networks.
OLSR in sparse networks

OLSR doesn’t work well in sparse networks:

- OLSR uses shortest hop routing:
  - depending on how it is tuned, either
    * the network loses connectivity; or
    * OLSR chooses marginal links.

- link state databases get desynchronised
  - transient routing loops.
OLSR routing loop example

A uses the direct route to S
B goes through A

A switches to the route through B before B has switched to the direct route

This situation will persist until the topology change is successfully flooded to B.

With Babel, A will delay switching routes until it can be sure that B has switched to the direct route.
Tuning example

OLSR in the community network *Funkfeuer*:

OLSR modified to take into account link quality: OLSR-ETX.

To avoid desynchronisation:

- MPR redundancy = 9,
- Hello interval = 5 s,
- TC interval = 2 s.

(No, this is not a typo.)
Desirable properties of a routing protocol

1. Decentralised (“survives a nuclear war”).

2. No pathologies
   - routing loops,
   - black holes
   not even transitory.

3. Converges fast to an acceptable configuration.

4. Eventually converges to an optimal configuration.
Babel: neighbour detection

A Babel node $A$ broadcasts more or less periodically

$$\text{Hello}(\text{seqno}, \text{interval})$$

a Hello packet decorated with a sequence number and the interval before the next hello.

The loss rate can be determined robustly, even when hello intervals are different or variable.

To each neighbour $B$, a node $A$ sends

$$\text{IHU}(B, \text{rate})$$

an I Heard You packet containing the link quality in the reverse direction
Babel: link quality

For each neighbour $B$, a node $A$ knows

- the rate $\alpha$ of success from $A$ to $B$ (IHU),
- the rate $\beta$ of success from $B$ to $A$ (Hello).

\[ A \xleftarrow{\alpha} \xrightarrow{\beta} B \]

The current implementation uses the ETX metric on wireless links:

\[ c_{AB} = \frac{1}{\alpha \beta} \]

On wired links, Babel uses the 2-3 metric:

\[ c_{AB} = \begin{cases} 1 & \text{if 2 of the last 3 were transmitted} \\ \infty & \text{otherwise} \end{cases} \]
Categories of routing protocols

1. Link state protocols
The global topology is flooded throughout the network. Every node independently computes the tree of shortest paths. Converge fast to optimality. No guarantee in case of desynchronisation.

2. Distance vector protocols
Every node locally computes its next hop. Optimal paths. Counting to infinity.

3. Link reversal protocols
Maintain locally a connected graph. Avoid loops. Non-optimal paths.
The Bellman-Ford algorithm

For each node \( X \), we maintain

\[
\begin{align*}
\text{d}(X) & \quad \text{distance to the source} \\
\text{nh}(X) & \quad \text{chosen successor}
\end{align*}
\]

Initially,

\[
\begin{align*}
\text{d}(S) & = 0 & \text{d}(X) & = \infty \\
\text{nh}(S) & = \perp & \text{nh}(X) & = \perp
\end{align*}
\]

At every step,

\[
\begin{align*}
\text{d}(X) & := \min_{Y \in \mathcal{V}(X)} c_{XY} + \text{d}(Y) \\
\text{nh}(X) & := \text{the neighbour that realises the min.}
\end{align*}
\]
Distributed Bellman-Ford

BF is robust: it can be iterated asynchronously.

Initially, \( d(S) = 0 \) \( d(X) = \infty \).

Often enough, \( Y \) broadcasts \( d(Y) \) to its neighbours.

When \( X \) receives \( d(Y) \),

- if \( \text{nh}(X) = Y \),
  \[ d(X) := c_{XY} + d(Y) \]

- if \( c_{XY} + d(Y) < d(X) \)
  \[ d(X) := c_{XY} + d(Y) \quad \text{nh}(X) := Y \]

If \( \text{nh}(X) = Y \), and \( Y \) stops broadcasting (timeout),

\[ d(X) := \infty \quad \text{nh}(X) := \bot \]
Distributed BF: convergence

Converges in $O(\Delta)$. 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$\infty$</td>
<td>1, nh = S</td>
<td>1, nh = S</td>
<td>1, nh = S</td>
<td>1, nh = S</td>
</tr>
<tr>
<td>B</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>2, nh = A</td>
<td>2, nh = A</td>
<td>2, nh = A</td>
</tr>
<tr>
<td>C</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>2, nh = A</td>
<td>2, nh = A</td>
<td>2, nh = A</td>
</tr>
</tbody>
</table>
Distributed BF: counting to infinity

Converges in $O(\infty)$. (In RIP, $\infty = 16$.)
Before convergence, routing loop.
« Good news travel fast, bad news travel forever. »
**BF: feasibility condition**

BF is **robust**, one can **refuse** some updates if they risk creating a loop.

When $X$ receives $(d(Y), f)$,

- if $nh(X) = Y$ and $\text{feasible}(Y, d(Y), f)$

  $$d(X) := c_{XY} + d(Y)$$

- if $c_{XY} + d(Y) < d(X)$ and $\text{feasible}(Y, d(Y), f)$

  $$d(X) := c_{XY} + d(Y)$$
  $$nh(X) := Y$$

where $\text{feasible}$ is some function that guarantees absence of loops.
Feasibility conditions

**BGP, Path Vector:**
\( f \) is the full path;
feasible(\( f \)) = \text{self} \not\in f.

**DSDV, AODV:**
\( \text{feasible}(d) \equiv c + d \leq d(\text{self}) \)
Invariants: \( d(X) \downarrow \) and if \( A \leftarrow B \) then \( d(A) < d(B) \).

**EIGRP/DUAL, Babel:**
We maintain \( \text{rd}(X) = \min_{t \leq \text{now}} d(X, t) \).
\( \text{feasible}(d) \equiv d < \text{rd}(\text{self}) \)
Invariants: \( \text{rd}(X) \downarrow \) and if \( A \leftarrow B \) then \( \text{rd}(A) < \text{rd}(B) \).


**Feasibility: example**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1, rd = 1</td>
<td>∞, rd = 1</td>
<td>∞, rd = 1</td>
<td>∞, rd = 1</td>
</tr>
<tr>
<td>B</td>
<td>2, rd = 2</td>
<td>2, rd = 2</td>
<td>∞, rd = 2</td>
<td>∞, rd = 2</td>
</tr>
<tr>
<td>C</td>
<td>2, rd = 2</td>
<td>2, rd = 2</td>
<td>∞, rd = 2</td>
<td>∞, rd = 2</td>
</tr>
</tbody>
</table>

Convergence in $O(\Delta)$. 
Feasibility: starvation

Feasibility conditions (2) and (3) cause starvation.

- $d(A) = 1$, $\text{rd}(A) = 1$
- $d(B) = 1$, $\text{rd}(B) = 1$

The only available route is not feasible.
Solving starvation

Idea: when no route is feasible, **reboot the whole network**.

DUAL/EIGRP performs a **global synchronisation** of all the routes to $S$.

DSDV, AODV and Babel use **sequenced routes**.
Solving starvation: sequenced routes

Route announcements are decorated with a sequence number:

\[(s, d(B))\]

where \(s \in \mathbb{N}\) is periodically incremented by the source:

\[d(S) = (s, 0) \quad (s \uparrow)\]
\[c + (s, m) = (s, c + m)\]

Define

\[(s, m) \leq (s', m') \quad \text{when} \quad s > s' \quad \text{or} \quad s = s' \text{ and } m \leq m'\]

\[\text{feasible}(s, m) \equiv (s, m) < \text{rd}.\]
## Sequenced routes: example

![Diagram of nodes S, A, B with sequenced routes]

<table>
<thead>
<tr>
<th></th>
<th>(1, 0)</th>
<th>(2, 0)</th>
<th>(2, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td>( \infty, \text{rd} = (1, 1) )</td>
<td>( \infty, \text{rd} = (1, 1) )</td>
<td>( (2, 2), \text{rd} = (2, 2) )</td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>( (1, 1), \text{rd} = (1, 1) )</td>
<td>( (2, 1), \text{rd} = (2, 1) )</td>
<td>( (2, 1), \text{rd} = (2, 1) )</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>( (1, 1), \text{rd} = (1, 1) )</td>
<td>( (2, 1), \text{rd} = (2, 1) )</td>
<td>( (2, 1), \text{rd} = (2, 1) )</td>
</tr>
</tbody>
</table>
Temporary starvation

S

A

B

d(S) = (1, 0)
d(B) = (1, 1)
d(A) = ∞    rd(A) = (1, 1)

A must wait for S to generate a new seqno, and for the new seqno to be propagated. This can take a significant amount of time.
Solving temporary starvation

When a Babel node suffers from temporary starvation, (routes available but not feasible) it sends an explicit request for a new seqno.

Unlike what happens in AODV, this request is not broadcast, which avoids an increasing diameter search — hop count and duplicate suppression is enough.
Multiple gateways

We want to have multiple Babel nodes announcing the same prefix without the need to synchronise sequence numbers.

Babel makes a distinction between source and destination.

A Babel announcement contains a triple

\[(s, d, id)\]

where \(id\) uniquely identifies the originator of this route. Reference distances are maintained per source and destination.
Multiple gateways: routing loops

With multiple gateways, Babel no longer guarantees the absence of loops.

\[
S_1 \quad A \quad B \quad S_2
\]

\[
d(A) = (17, 1) \quad d(B) = (43, 1)
\]

\[
rd(A, S_1) = (17, 1) \quad rd(B, S_2) = (43, 1)
\]

We do guarantee that a loop \textit{disappers in } \(O(n)\), where \(n\) is the size of the loop.
Overlapping prefixes

A routing loop may also appear because of overlapping prefixes:

0.0.0.0/0 ——— A ——— B ——— C

The link between B and C disappears:

0.0.0.0/0 ——— A ——— B ——— C

If B switches to A, there is a temporary routing loop. This can only happen after a retraction. Babel obeys a hold time after a route retraction. This is the least satisfactory part of Babel.
Completed work

• Full implementation of the Babel protocol;
• for Linux, but designed to be portable;
• deployed on a few nodes (OpenWRT + Debian);
• rich redistribution and filtering language.

Work in progress:

• Minor but incompatible revisions to the protocol;
• GUI (Pejman Attar and Alex Roso);
• simulation (Yoann Canal and Łukasz Fronc);
• Porting under Mac OS X (Grégoire Henry).
Open problems

- Deployment,
- deployment,
- deployment.

- Tune the **heuristics** (requests, triggered updates).
- **Hybrid metrics** (diversity, delay, battery etc.).
- Porting under Windows (Mac OS X in progress).
- **cross-layer triggers** ?
Conclusion

Babel is a routing protocol designed for sparse networks.

- routing according to link quality;
- robust in case of desynchronisation;
- rapid reconvergence after a mobility event.

It needs to be tested in large networks.

http://www.pps.jussieu.fr/~jch/software/babel/
http://wifi.pps.jussieu.fr/