The Babel Routing Protocol

Abstract

   Babel is a loop-avoiding distance-vector routing protocol that is
   robust and efficient both in ordinary wired networks and in wireless
   mesh networks.

Status of This Memo

   This document is not an Internet Standards Track specification; it is
   published for examination, experimental implementation, and
   evaluation.

   This document defines an Experimental Protocol for the Internet
   community. This is a contribution to the RFC Series, independently
   of any other RFC stream. The RFC Editor has chosen to publish this
   document at its discretion and makes no statement about its value for
   implementation or deployment. Documents approved for publication by
   the RFC Editor are not a candidate for any level of Internet
   Standard; see Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc6126.

Table of Contents

1. Introduction

   Babel is a loop-avoiding distance-vector routing protocol that is
   designed to be robust and efficient both in networks using prefix-
   based routing and in networks using flat routing ("mesh networks"),
   and both in relatively stable wired networks and in highly dynamic
   wireless networks.

1.1. Features

   The main property that makes Babel suitable for unstable networks is
   that, unlike naive distance-vector routing protocols [RIP], it
   strongly limits the frequency and duration of routing pathologies
   such as routing loops and black-holes during reconvergence. Even
   after a mobility event is detected, a Babel network usually remains
   loop-free. Babel then quickly reconverges to a configuration that
   preserves the loop-freedom and connectedness of the network, but is
   not necessarily optimal; in many cases, this operation requires no
   packet exchanges at all. Babel then slowly converges, in a time on
   the scale of minutes, to an optimal configuration. This is achieved
   by using sequenced routes, a technique pioneered by Destination-
   Sequenced Distance-Vector routing [DSDV].

   More precisely, Babel has the following properties:

   o   when every prefix is originated by at most one router, Babel never
       suffers from routing loops;

   o   when a prefix is originated by multiple routers, Babel may
       occasionally create a transient routing loop for this particular
       prefix; this loop disappears in a time proportional to its
       diameter, and never again (up to an arbitrary garbage-collection
       (GC) time) will the routers involved participate in a routing loop
       for the same prefix;

   o   assuming reasonable packet loss rates, any routing black-holes
       that may appear after a mobility event are corrected in a time at
       most proportional to the network's diameter.

   Babel has provisions for link quality estimation and for fairly
   arbitrary metrics. When configured suitably, Babel can implement
   shortest-path routing, or it may use a metric based, for example, on
   measured packet loss.

   Babel nodes will successfully establish an association even when they
   are configured with different parameters. For example, a mobile node
   that is low on battery may choose to use larger time constants (hello
   and update intervals, etc.) than a node that has access to wall

power. Conversely, a node that detects high levels of mobility may
choose to use smaller time constants. The ability to build such
heterogeneous networks makes Babel particularly adapted to the
wireless environment.

Finally, Babel is a hybrid routing protocol, in the sense that it can
carry routes for multiple network-layer protocols (IPv4 and IPv6),
whichever protocol the Babel packets are themselves being carried
over.

1.2. Limitations

Babel has two limitations that make it unsuitable for use in some
environments. First, Babel relies on periodic routing table updates
rather than using a reliable transport; hence, in large, stable
networks it generates more traffic than protocols that only send
updates when the network topology changes. In such networks,
protocols such as OSPF [OSPF], IS-IS [IS-IS], or the Enhanced
Interior Gateway Routing Protocol (EIGRP) [EIGRP] might be more
suitable.

Second, Babel does impose a hold time when a prefix is retracted
(Section 3.5.5). While this hold time does not apply to the exact
prefix being retracted, and hence does not prevent fast reconvergence
should it become available again, it does apply to any shorter prefix
that covers it. Hence, if a previously deaggregated prefix becomes
aggregated, it will be unreachable for a few minutes. This makes
Babel unsuitable for use in mobile networks that implement automatic
prefix aggregation.

1.3. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

2. Conceptual Description of the Protocol

Babel is a mostly loop-free distance vector protocol: it is based on
the Bellman-Ford protocol, just like the venerable RIP [RIP], but
includes a number of refinements that either prevent loop formation
altogether, or ensure that a loop disappears in a timely manner and
doesn't form again.

Conceptually, Bellman-Ford is executed in parallel for every source
of routing information (destination of data traffic). In the
following discussion, we fix a source S; the reader will recall that
the same algorithm is executed for all sources.

2.1. Costs, Metrics, and Neighbourship

   As many routing algorithms, Babel computes costs of links between any
   two neighbouring nodes, abstract values attached to the edges between
   two nodes. We write C(A, B) for the cost of the edge from node A to
   node B.

   Given a route between any two nodes, the metric of the route is the
   sum of the costs of all the edges along the route. The goal of the
   routing algorithm is to compute, for every source S, the tree of the
   routes of lowest metric to S.

   Costs and metrics need not be integers. In general, they can be
   values in any algebra that satisfies two fairly general conditions
   (Section 3.5.2).

   A Babel node periodically broadcasts Hello messages to all of its
   neighbours; it also periodically sends an IHU ("I Heard You") message
   to every neighbour from which it has recently heard a Hello. From
   the information derived from Hello and IHU messages received from its
   neighbour B, a node A computes the cost C(A, B) of the link from A to
   B.

2.2. The Bellman-Ford Algorithm

   Every node A maintains two pieces of data: its estimated distance to
   S, written D(A), and its next-hop router to S, written NH(A).
   Initially, D(S) = 0, D(A) is infinite, and NH(A) is undefined.

   Periodically, every node B sends to all of its neighbours a route
   update, a message containing D(B). When a neighbour A of B receives
   the route update, it checks whether B is its selected next hop; if
   that is the case, then NH(A) is set to B, and D(A) is set to C(A, B)
   + D(B). If that is not the case, then A compares C(A, B) + D(B) to
   its current value of D(A). If that value is smaller, meaning that
   the received update advertises a route that is better than the
   currently selected route, then NH(A) is set to B, and D(A) is set to
   C(A, B) + D(B).

   A number of refinements to this algorithm are possible, and are used
   by Babel. In particular, convergence speed may be increased by
   sending unscheduled "triggered updates" whenever a major change in
   the topology is detected, in addition to the regular, scheduled
   updates. Additionally, a node may maintain a number of alternate
   routes, which are being advertised by neighbours other than its
   selected neighbour, and which can be used immediately if the selected
   route were to fail.

2.3. Transient Loops in Bellman-Ford

   It is well known that a naive application of Bellman-Ford to
   distributed routing can cause transient loops after a topology
   change. Consider for example the following diagram:

```
        B
     1 /|
    1  / |
 S --- A  |1
        \ |
       1 \|
          C
```
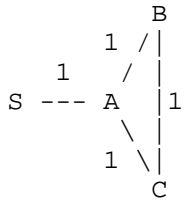
   After convergence, D(B) = D(C) = 2, with NH(B) = NH(C) = A.

   Suppose now that the link between S and A fails:

```
        B
     1 /|
       / |
 S     A  |1
        \ |
       1 \|
          C
```
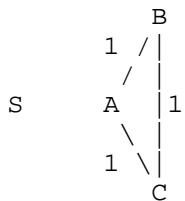
   When it detects the failure of the link, A switches its next hop to B
   (which is still advertising a route to S with metric 2), and
   advertises a metric equal to 3, and then advertises a new route with
   metric 3. This process of nodes changing selected neighbours and
   increasing their metric continues until the advertised metric reaches
   "infinity", a value larger than all the metrics that the routing
   protocol is able to carry.

2.4. Feasibility Conditions

   Bellman-Ford is a very robust algorithm: its convergence properties
   are preserved when routers delay route acquisition or when they
   discard some updates. Babel routers discard received route
   announcements unless they can prove that accepting them cannot
   possibly cause a routing loop.

   More formally, we define a condition over route announcements, known
   as the feasibility condition, that guarantees the absence of routing
   loops whenever all routers ignore route updates that do not satisfy
   the feasibility condition. In effect, this makes Bellman-Ford into a
   family of routing algorithms, parameterised by the feasibility
   condition.

Many different feasibility conditions are possible. For example, BGP
can be modelled as being a distance-vector protocol with a (rather
drastic) feasibility condition: a routing update is only accepted
when the receiving node's AS number is not included in the update's
AS-Path attribute (note that BGP's feasibility condition does not
ensure the absence of transitory "micro-loops" during reconvergence).

Another simple feasibility condition, used in Destination-Sequenced
Distance-Vector (DSDV) routing [DSDV] and in Ad hoc On-Demand
Distance Vector (AODV) routing, stems from the following observation:
a routing loop can only arise after a router has switched to a route
with a larger metric than the route that it had previously selected.
Hence, one could decide that a route is feasible only when its metric
at the local node would be no larger than the metric of the currently
selected route, i.e., an announcement carrying a metric $D(B)$ is
accepted by A when $C(A, B) + D(B) <= D(A)$. If all routers obey this
constraint, then the metric at every router is nonincreasing, and the
following invariant is always preserved: if A has selected B as its
successor, then $D(B) < D(A)$, which implies that the forwarding graph
is loop-free.

Babel uses a slightly more refined feasibility condition, used in
EIGRP [DUAL]. Given a router A, define the feasibility distance of
A, written $FD(A)$, as the smallest metric that A has ever advertised
for S to any of its neighbours. An update sent by a neighbour B of A
is feasible when the metric $D(B)$ advertised by B is strictly smaller
than A's feasibility distance, i.e., when $D(B) < FD(A)$.

It is easy to see that this latter condition is no more restrictive
than DSDV-feasibility. Suppose that node A obeys DSDV-feasibility;
then $D(A)$ is nonincreasing, hence at all times $D(A) <= FD(A)$.
Suppose now that A receives a DSDV-feasible update that advertises a
metric $D(B)$. Since the update is DSDV-feasible, $C(A, B) + D(B) <=
D(A)$, hence $D(B) < D(A)$, and since $D(A) <= FD(A)$, $D(B) < FD(A)$.

To see that it is strictly less restrictive, consider the following
diagram, where A has selected the route through B, and $D(A) = FD(A) =
2$. Since $D(C) = 1 < FD(A)$, the alternate route through C is feasible
for A, although its metric $C(A, C) + D(C) = 5$ is larger than that of
the currently selected route:

```
    B
1 / \ 1
  /    \
S      A
 \    /
1 \ / 4
   C
```
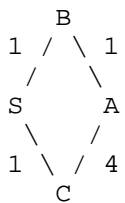
To show that this feasibility condition still guarantees loop-
freedom, recall that at the time when A accepts an update from B, the
metric D(B) announced by B is no smaller than FD(B); since it is
smaller than FD(A), at that point in time FD(B) < FD(A). Since this
property is preserved when A sends updates, it remains true at all
times, which ensures that the forwarding graph has no loops.

2.5. Solving Starvation: Sequencing Routes

Obviously, the feasibility conditions defined above cause starvation
when a router runs out of feasible routes. Consider the following
diagram, where both A and B have selected the direct route to S:

```
    A
1 /|          D(A) = 1
 / |          FD(A) = 1
S  |1
 \ |          D(B) = 2
2 \|          FD(B) = 2
    B
```
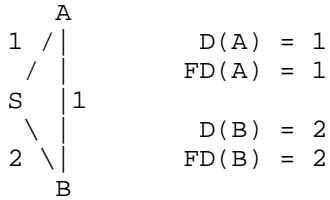
Suppose now that the link between A and S breaks:

```
    A
    |
    |          FD(A) = 1
S  |1
 \ |          D(B) = 2
2 \|          FD(B) = 2
    B
```
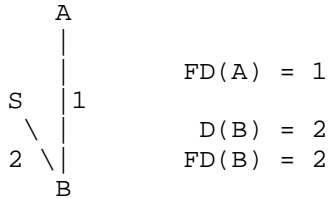
The only route available from A to S, the one that goes through B, is
not feasible: A suffers from a spurious starvation.

At this point, the whole network must be rebooted in order to solve
the starvation; this is essentially what EIGRP does when it performs
a global synchronisation of all the routers in the network with the
source (the "active" phase of EIGRP).

Babel reacts to starvation in a less drastic manner, by using
sequenced routes, a technique introduced by DSDV and adopted by AODV.
In addition to a metric, every route carries a sequence number, a
nondecreasing integer that is propagated unchanged through the
network and is only ever incremented by the source; a pair (s, m),
where s is a sequence number and m a metric, is called a distance.

A received update is feasible when either it is more recent than the
feasibility distance maintained by the receiving node, or it is

equally recent and the metric is strictly smaller. More formally, if
FD(A) = (s, m), then an update carrying the distance (s', m') is
feasible when either s' > s, or s = s' and m' < m.

Assuming the sequence number of S is 137, the diagram above becomes:

```
    A
    |
    |         FD(A) = (137, 1)
S  |1
  \ |          D(B) = (137, 2)
2 \|         FD(B) = (137, 2)
    B
```
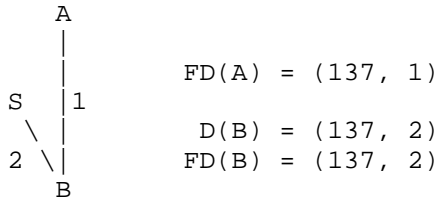
After S increases its sequence number, and the new sequence number is
propagated to B, we have:

```
    A
    |
    |         FD(A) = (137, 1)
S  |1
  \ |          D(B) = (138, 2)
2 \|         FD(B) = (138, 2)
    B
```
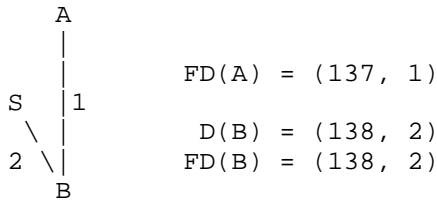
at which point the route through B becomes feasible again.

Note that while sequence numbers are used for determining
feasibility, they are not necessarily used in route selection: a node
will normally ignore the sequence number when selecting a route
(Section 3.6).

2.6. Requests

   In DSDV, the sequence number of a source is increased periodically.
   A route becomes feasible again after the source increases its
   sequence number, and the new sequence number is propagated through
   the network, which may, in general, require a significant amount of
   time.

   Babel takes a different approach. When a node detects that it is
   suffering from a potentially spurious starvation, it sends an
   explicit request to the source for a new sequence number. This
   request is forwarded hop by hop to the source, with no regard to the
   feasibility condition. Upon receiving the request, the source
   increases its sequence number and broadcasts an update, which is
   forwarded to the requesting node.

Note that after a change in network topology not all such requests
will, in general, reach the source, as some will be sent over links
that are now broken. However, if the network is still connected,
then at least one among the nodes suffering from spurious starvation
has an (unfeasible) route to the source; hence, in the absence of
packet loss, at least one such request will reach the source.
(Resending requests a small number of times compensates for packet
loss.)

Since requests are forwarded with no regard to the feasibility
condition, they may, in general, be caught in a forwarding loop; this
is avoided by having nodes perform duplicate detection for the
requests that they forward.

2.7. Multiple Routers

The above discussion assumes that every prefix is originated by a
single router. In real networks, however, it is often necessary to
have a single prefix originated by multiple routers; for example, the
default route will be originated by all of the edge routers of a
routing domain.

Since synchronising sequence numbers between distinct routers is
problematic, Babel treats routes for the same prefix as distinct
entities when they are originated by different routers: every route
announcement carries the router-id of its originating router, and
feasibility distances are not maintained per prefix, but per source,
where a source is a pair of a router-id and a prefix. In effect,
Babel guarantees loop-freedom for the forwarding graph to every
source; since the union of multiple acyclic graphs is not in general
acyclic, Babel does not in general guarantee loop-freedom when a
prefix is originated by multiple routers, but any loops will be
broken in a time at most proportional to the diameter of the loop --
as soon as an update has "gone around" the routing loop.

Consider for example the following diagram, where A has selected the
default route through S, and B has selected the one through S':

```
        1       1       1
::/0 -- S --- A --- B --- S' -- ::/0
```

Suppose that both default routes fail at the same time; then nothing
prevents A from switching to B, and B simultaneously switching to A.
However, as soon as A has successfully advertised the new route to B,
the route through A will become unfeasible for B. Conversely, as
soon as B will have advertised the route through A, the route through
B will become unfeasible for A.

In effect, the routing loop disappears at the latest when routing
information has gone around the loop. Since this process can be
delayed by lost packets, Babel makes certain efforts to ensure that
updates are sent reliably after a router-id change.

Additionally, after the routers have advertised the two routes, both
sources will be in their source tables, which will prevent them from
ever again participating in a routing loop involving routes from S
and S' (up to the source GC time, which, available memory permitting,
can be set to arbitrarily large values).

2.8. Overlapping Prefixes

In the above discussion, we have assumed that all prefixes are
disjoint, as is the case in flat ("mesh") routing. In practice,
however, prefixes may overlap: for example, the default route
overlaps with all of the routes present in the network.

After a route fails, it is not correct in general to switch to a
route that subsumes the failed route. Consider for example the
following configuration:

```
           1      1
    ::/0 -- A --- B --- C
```

Suppose that node C fails. If B forwards packets destined to C by
following the default route, a routing loop will form, and persist
until A learns of B's retraction of the direct route to C. Babel
avoids this pitfall by maintaining an "unreachable" route for a few
minutes after a route is retracted; the time for which such a route
must be maintained should be the worst-case propagation time of the
retraction of the route to C.

3. Protocol Operation

Every Babel speaker is assigned a router-id, which is an arbitrary
string of 8 octets that is assumed unique across the routing domain.
We suggest that router-ids should be assigned in modified EUI-64
format [ADDRARCH]. (As a matter of fact, the protocol encoding is
slightly more compact when router-ids are assigned in the same manner
as the IPv6 layer assigns host IDs.)

3.1. Message Transmission and Reception

Babel protocol packets are sent in the body of a UDP datagram. Each
Babel packet consists of one or more TLVs.

The source address of a Babel packet is always a unicast address,
link-local in the case of IPv6. Babel packets may be sent to a well-
known (link-local) multicast address (this is the usual case) or to a
(link-local) unicast address. In normal operation, a Babel speaker
sends both multicast and unicast packets to its neighbours.

With the exception of Hello TLVs and acknowledgements, all Babel TLVs
can be sent to either unicast or multicast addresses, and their
semantics does not depend on whether the destination was a unicast or
multicast address. Hence, a Babel speaker does not need to determine
the destination address of a packet that it receives in order to
interpret it.

A moderate amount of jitter is applied to packets sent by a Babel
speaker: outgoing TLVs are buffered and SHOULD be sent with a small
random delay. This is done for two purposes: it avoids
synchronisation of multiple Babel speakers across a network [JITTER],
and it allows for the aggregation of multiple TLVs into a single
packet.

The exact delay and amount of jitter applied to a packet depends on
whether it contains any urgent TLVs. Acknowledgement TLVs MUST be
sent before the deadline specified in the corresponding request. The
particular class of updates specified in Section 3.7.2 MUST be sent
in a timely manner. The particular class of request and update TLVs
specified in Section 3.8.2 SHOULD be sent in a timely manner.

3.2. Data Structures

Every Babel speaker maintains a number of data structures.

3.2.1. Sequence Number

A node's sequence number is a 16-bit integer that is included in
route updates sent for routes originated by this node. A node
increments its sequence number (modulo 2^{16}) whenever it receives a
request for a new sequence number (Section 3.8.1.2).

A node SHOULD NOT increment its sequence number (seqno)
spontaneously, since increasing seqnos makes it less likely that
other nodes will have feasible alternate routes when their selected
routes fail.

3.2.2. The Interface Table

The interface table contains the list of interfaces on which the node
speaks the Babel protocol. Every interface table entry contains the
interface's Hello seqno, a 16-bit integer that is sent with each

Hello TLV on this interface and is incremented (modulo 2^16) whenever
a Hello is sent. (Note that an interface's Hello seqno is unrelated
to the node's seqno.)

There are two timers associated with each interface table entry --
the Hello timer, which governs the sending of periodic Hello and IHU
packets, and the update timer, which governs the sending of periodic
route updates.

3.2.3. The Neighbour Table

The neighbour table contains the list of all neighbouring interfaces
from which a Babel packet has been recently received. The neighbour
table is indexed by pairs of the form (interface, address), and every
neighbour table entry contains the following data:

o the local node's interface over which this neighbour is reachable;

o the address of the neighbouring interface;

o a history of recently received Hello packets from this neighbour;
   this can, for example, be a sequence of n bits, for some small
   value n, indicating which of the n hellos most recently sent by
   this neighbour have been received by the local node;

o the "transmission cost" value from the last IHU packet received
   from this neighbour, or FFFF hexadecimal (infinity) if the IHU
   hold timer for this neighbour has expired;

o the neighbour's expected Hello sequence number, an integer modulo
   2^16.

There are two timers associated with each neighbour entry -- the
hello timer, which is initialised from the interval value carried by
Hello TLVs, and the IHU timer, which is initialised to a small
multiple of the interval carried in IHU TLVs.

Note that the neighbour table is indexed by IP addresses, not by
router-ids: neighbourship is a relationship between interfaces, not
between nodes. Therefore, two nodes with multiple interfaces can
participate in multiple neighbourship relationships, a fairly common
situation when wireless nodes with multiple radios are involved.

3.2.4. The Source Table

The source table is used to record feasibility distances. It is
indexed by triples of the form (prefix, plen, router-id), and every
source table entry contains the following data:

o the prefix (prefix, plen), where plen is the prefix length, that
   this entry applies to;

o the router-id of a router originating this prefix;

o a pair (seqno, metric), this source's feasibility distance.

There is one timer associated with each entry in the source table --
the source garbage-collection timer. It is initialised to a time on
the order of minutes and reset as specified in Section 3.7.3.

3.2.5. The Route Table

The route table contains the routes known to this node. It is
indexed by triples of the form (prefix, plen, neighbour), and every
route table entry contains the following data:

o the source (prefix, plen, router-id) for which this route is
   advertised;

o the neighbour that advertised this route;

o the metric with which this route was advertised by the neighbour,
   or FFFF hexadecimal (infinity) for a recently retracted route;

o the sequence number with which this route was advertised;

o the next-hop address of this route;

o a boolean flag indicating whether this route is selected, i.e.,
   whether it is currently being used for forwarding and is being
   advertised.

There is one timer associated with each route table entry -- the
route expiry timer. It is initialised and reset as specified in
Section 3.5.4.

3.2.6. The Table of Pending Requests

The table of pending requests contains a list of seqno requests that
the local node has sent (either because they have been originated
locally, or because they were forwarded) and to which no reply has
been received yet. This table is indexed by prefixes, and every
entry in this table contains the following data:

o the prefix, router-id, and seqno being requested;

   o the neighbour, if any, on behalf of which we are forwarding this
      request;

   o a small integer indicating the number of times that this request
      will be resent if it remains unsatisfied.

   There is one timer associated with each pending request; it governs
   both the resending of requests and their expiry.

3.3. Acknowledged Packets

   A Babel speaker may request that any neighbour receiving a given
   packet reply with an explicit acknowledgement within a given time.
   While the use of acknowledgement requests is optional, every Babel
   speaker MUST be able to reply to such a request.

   An acknowledgement MUST be sent to a unicast destination. On the
   other hand, acknowledgement requests may be sent to either unicast or
   multicast destinations, in which case they request an acknowledgement
   from all of the receiving nodes.

   When to request acknowledgements is a matter of local policy; the
   simplest strategy is to never request acknowledgements and to rely on
   periodic updates to ensure that any reachable routes are eventually
   propagated throughout the routing domain. For increased efficiency,
   we suggest that acknowledged packets should be used in order to send
   urgent updates (Section 3.7.2) when the number of neighbours on a
   given interface is small. Since Babel is designed to deal gracefully
   with packet loss on unreliable media, sending all packets with
   acknowledgement requests is not necessary, and not even recommended,
   as the acknowledgements cause additional traffic and may force
   additional Address Resolution Protocol (ARP) or Neighbour Discovery
   exchanges.

3.4. Neighbour Acquisition

   Neighbour acquisition is the process by which a Babel node discovers
   the set of neighbours heard over each of its interfaces and
   ascertains bidirectional reachability. On unreliable media,
   neighbour acquisition additionally provides some statistics that MAY
   be used in link quality computation.

3.4.1. Reverse Reachability Detection

   Every Babel node sends periodic Hellos over each of its interfaces.
   Each Hello TLV carries an increasing (modulo 2^{16}) sequence number
   and the interval between successive periodic packets sent on this
   particular interface.

In addition to the periodic Hello packets, a node MAY send
unscheduled Hello packets, e.g., to accelerate link cost estimation
when a new neighbour is discovered, or when link conditions have
suddenly changed.

A node MAY change its Hello interval. The Hello interval MAY be
decreased at any time; it SHOULD NOT be increased, except immediately
before sending a Hello packet. (Equivalently, a node SHOULD send an
unscheduled Hello immediately after increasing its Hello interval.)

How to deal with received Hello TLVs and what statistics to maintain
are considered local implementation matters; typically, a node will
maintain some sort of history of recently received Hellos. A
possible algorithm is described in Appendix A.1.

After receiving a Hello, or determining that it has missed one, the
node recomputes the association's cost (Section 3.4.3) and runs the
route selection procedure (Section 3.6).

3.4.2. Bidirectional Reachability Detection

In order to establish bidirectional reachability, every node sends
periodic IHU ("I Heard You") TLVs to each of its neighbours. Since
IHUs carry an explicit interval value, they MAY be sent less often
than Hellos in order to reduce the amount of routing traffic in dense
networks; in particular, they SHOULD be sent less often than Hellos
over links with little packet loss. While IHUs are conceptually
unicast, they SHOULD be sent to a multicast address in order to avoid
an ARP or Neighbour Discovery exchange and to aggregate multiple IHUs
in a single packet.

In addition to the periodic IHUs, a node MAY, at any time, send an
unscheduled IHU packet. It MAY also, at any time, decrease its IHU
interval, and it MAY increase its IHU interval immediately before
sending an IHU.

Every IHU TLV contains two pieces of data: the link's rxcost
(reception cost) from the sender's perspective, used by the neighbour
for computing link costs (Section 3.4.3), and the interval between
periodic IHU packets. A node receiving an IHU updates the value of
the sending neighbour's txcost (transmission cost), from its
perspective, to the value contained in the IHU, and resets this
neighbour's IHU timer to a small multiple of the value received in
the IHU.

When a neighbour's IHU timer expires, its txcost is set to infinity.

After updating a neighbour's txcost, the receiving node recomputes
the neighbour's cost (Section 3.4.3) and runs the route selection
procedure (Section 3.6).

3.4.3. Cost Computation

A neighbourship association's link cost is computed from the values
maintained in the neighbour table -- namely, the statistics kept in
the neighbour table about the reception of Hellos, and the txcost
computed from received IHU packets.

For every neighbour, a Babel node computes a value known as this
neighbour's rxcost. This value is usually derived from the Hello
history, which may be combined with other data, such as statistics
maintained by the link layer. The rxcost is sent to a neighbour in
each IHU.

How the txcost and rxcost are combined in order to compute a link's
cost is a matter of local policy; as far as Babel's correctness is
concerned, only the following conditions MUST be satisfied:

o the cost is strictly positive;

o if no hellos were received recently, then the cost is infinite;

o if the txcost is infinite, then the cost is infinite.

Note that while this document does not constrain cost computation any
further, not all cost computation strategies will give good results.
We give a few examples of strategies for computing a link's cost that
are known to work well in practice in Appendix A.2.

3.5. Routing Table Maintenance

Conceptually, a Babel update is a quintuple (prefix, plen, router-id,
seqno, metric), where (prefix, plen) is the prefix for which a route
is being advertised, router-id is the router-id of the router
originating this update, seqno is a nondecreasing (modulo 2^{16})
integer that carries the originating router seqno, and metric is the
announced metric.

Before being accepted, an update is checked against the feasibility
condition (Section 3.5.1), which ensures that the route does not
create a routing loop. If the feasibility condition is not
satisfied, the update is either ignored or treated as a retraction,
depending on some other conditions (Section 3.5.4). If the
feasibility condition is satisfied, then the update cannot possibly
cause a routing loop, and the update is accepted.

3.5.1. The Feasibility Condition

   The feasibility condition is applied to all received updates. The
   feasibility condition compares the metric in the received update with
   the metrics of the updates previously sent by the receiving node;
   updates with finite metrics large enough to cause a loop are
   discarded.

   A feasibility distance is a pair (seqno, metric), where seqno is an
   integer modulo 2^16 and metric is a positive integer. Feasibility
   distances are compared lexicographically, with the first component
   inverted: we say that a distance (seqno, metric) is strictly better
   than a distance (seqno', metric'), written

      (seqno, metric) < (seqno', metric')

   when

      seqno > seqno' or (seqno = seqno' and metric < metric')

   where sequence numbers are compared modulo 2^16.

   Given a source (p, plen, id), a node's feasibility distance for this
   source is the minimum, according to the ordering defined above, of
   the distances of all the finite updates ever sent by this particular
   node for the prefix (p, plen) carrying the router-id id. Feasibility
   distances are maintained in the source table; the exact procedure is
   given in Section 3.7.3.

   A received update is feasible when either it is a retraction (its
   metric is FFFF hexadecimal), or the advertised distance is strictly
   better, in the sense defined above, than the feasibility distance for
   the corresponding source. More precisely, a route advertisement
   carrying the quintuple (prefix, plen, router-id, seqno, metric) is
   feasible if one of the following conditions holds:

   o metric is infinite; or

   o no entry exists in the source table indexed by (id, prefix, plen);
      or

   o an entry (prefix, plen, router-id, seqno', metric') exists in the
      source table, and either

      * seqno' < seqno or

      * seqno = seqno' and metric < metric'.

   Note that the feasibility condition considers the metric advertised
   by the neighbour, not the route's metric; hence, a fluctuation in a
   neighbour's cost cannot render a selected route unfeasible.

3.5.2. Metric Computation

   A route's metric is computed from the metric advertised by the
   neighbour and the neighbour's link cost. Just like cost computation,
   metric computation is considered a local policy matter; as far as
   Babel is concerned, the function $M(c, m)$ used for computing a metric
   from a locally computed link cost and the metric advertised by a
   neighbour MUST only satisfy the following conditions:

   o if c is infinite, then $M(c, m)$ is infinite;

   o M is strictly monotonic: $M(c, m) > m$.

   Additionally, the metric SHOULD satisfy the following condition:

   o M is isotonic: if $m <= m'$, then $M(c, m) <= M(c, m')$.

   Note that while strict monotonicity is essential to the integrity of
   the network (persistent routing loops may appear if it is not
   satisfied), isotonicity is not: if it is not satisfied, Babel will
   still converge to a locally optimal routing table, but might not
   reach a global optimum (in fact, such a global optimum may not even
   exist).

   As with cost computation, not all strategies for computing route
   metrics will give good results. In particular, some metrics are more
   likely than others to lead to routing instabilities (route flapping).
   In Appendix A.3, we give a number of examples of strictly monotonic,
   isotonic routing metrics that are known to work well in practice.

3.5.3. Encoding of Updates

   In a large network, the bulk of Babel traffic consists of route
   updates; hence, some care has been given to encoding them
   efficiently. An Update TLV itself only contains the prefix, seqno,
   and metric, while the next hop is derived either from the network-
   layer source address of the packet or from an explicit Next Hop TLV
   in the same packet. The router-id is derived from a separate
   Router-Id TLV in the same packet, which optimises the case when
   multiple updates are sent with the same router-id.

   Additionally, a prefix of the advertised prefix can be omitted in an
   Update TLV, in which case it is copied from a previous Update TLV in
   the same packet -- this is known as address compression [PACKETBB].

Finally, as a special optimisation for the case when a router-id
coincides with the interface-id part of an IPv6 address, the
router-id can optionally be derived from the low-order bits of the
advertised prefix.

The encoding of updates is described in detail in Section 4.4.

3.5.4. Route Acquisition

When a Babel node receives an update (id, prefix, seqno, metric) from
a neighbour neigh with a link cost value equal to cost, it checks
whether it already has a routing table entry indexed by (neigh, id,
prefix).

If no such entry exists:

o if the update is unfeasible, it is ignored;

o if the metric is infinite (the update is a retraction), the update
   is ignored;

o otherwise, a new route table entry is created, indexed by (neigh,
   id, prefix), with seqno equal to seqno and an advertised metric
   equal to the metric carried by the update.

If such an entry exists:

o if the entry is currently installed and the update is unfeasible,
   then the behaviour depends on whether the router-ids of the two
   entries match. If the router-ids are different, the update is
   treated as though it were a retraction (i.e., as though the metric
   were FFFF hexadecimal). If the router-ids are equal, the update
   is ignored;

o otherwise (i.e., if either the update is feasible or the entry is
   not currently installed), then the entry's sequence number,
   advertised metric, metric, and router-id are updated and, unless
   the advertised metric is infinite, the route's expiry timer is
   reset to a small multiple of the Interval value included in the
   update.

When a route's expiry timer triggers, the behaviour depends on
whether the route's metric is finite. If the metric is finite, it is
set to infinity and the expiry timer is reset. If the metric is
already infinite, the route is flushed from the route table.

After the routing table is updated, the route selection procedure
(Section 3.6) is run.

3.5.5. Hold Time

   When a prefix p is retracted, because all routes are unfeasible, too
   old, or have an infinite metric, and a shorter prefix p' that covers
   p is reachable, p' cannot in general be used for routing packets
   destined to p without running the risk of creating a routing loop
   (Section 2.8).

   To avoid this issue, whenever a prefix is retracted, a routing table
   entry with infinite metric is maintained as described in
   Section 3.5.4 above, and packets destined for that prefix MUST NOT be
   forwarded by following a route for a shorter prefix. The infinite
   metric entry is maintained until it is superseded by a feasible
   update; if no such update arrives within the route hold time, the
   entry is flushed.

3.6. Route Selection

   Route selection is the process by which a single route for a given
   prefix is selected to be used for forwarding packets and to be
   re-advertised to a node's neighbours.

   Babel is designed to allow flexible route selection policies. As far
   as the protocol's correctness is concerned, the route selection
   policy MUST only satisfy the following properties:

   o a route with infinite metric (a retracted route) is never
      selected;

   o an unfeasible route is never selected.

   Note, however, that Babel does not naturally guarantee the stability
   of routing, and configuring conflicting route selection policies on
   different routers may lead to persistent route oscillation.

   Defining a good route selection policy for Babel is an open research
   problem. Route selection can take into account multiple mutually
   contradictory criteria; in roughly decreasing order of importance,
   these are:

   o routes with a small metric should be preferred over routes with a
      large metric;

   o switching router-ids should be avoided;

   o routes through stable neighbours should be preferred over routes
      through unstable ones;

   o stable routes should be preferred over unstable ones;

   o switching next hops should be avoided.

   A simple strategy is to choose the feasible route with the smallest
   metric, with a small amount of hysteresis applied to avoid switching
   router-ids.

   After the route selection procedure is run, triggered updates
   (Section 3.7.2) and requests (Section 3.8.2) are sent.

3.7. Sending Updates

   A Babel speaker advertises to its neighbours its set of selected
   routes. Normally, this is done by sending one or more multicast
   packets containing Update TLVs on all of its connected interfaces;
   however, on link technologies where multicast is significantly more
   expensive than unicast, a node MAY choose to send multiple copies of
   updates in unicast packets when the number of neighbours is small.

   Additionally, in order to ensure that any black-holes are reliably
   cleared in a timely manner, a Babel node sends retractions (updates
   with an infinite metric) for any recently retracted prefixes.

   If an update is for a route injected into the Babel domain by the
   local node (e.g., the address of a local interface, the prefix of a
   directly attached network, or redistributed from a different routing
   protocol), the router-id is set to the local id, the metric is set to
   some arbitrary finite value (typically 0), and the seqno is set to
   the local router's sequence number.

   If an update is for a route learned from another Babel speaker, the
   router-id and sequence number are copied from the routing table
   entry, and the metric is computed as specified in Section 3.5.2.

3.7.1. Periodic Updates

   Every Babel speaker periodically advertises all of its selected
   routes on all of its interfaces, including any recently retracted
   routes. Since Babel doesn't suffer from routing loops (there is no
   "counting to infinity") and relies heavily on triggered updates
   (Section 3.7.2), this full dump only needs to happen infrequently.

3.7.2. Triggered Updates

   In addition to the periodic routing updates, a Babel speaker sends
   unscheduled, or triggered, updates in order to inform its neighbours
   of a significant change in the network topology.

A change of router-id for the selected route to a given prefix may be
indicative of a routing loop in formation; hence, a node MUST send a
triggered update in a timely manner whenever it changes the selected
router-id for a given destination. Additionally, it SHOULD make a
reasonable attempt at ensuring that all neighbours receive this
update.

There are two strategies for ensuring that. If the number of
neighbours is small, then it is reasonable to send the update
together with an acknowledgement request; the update is resent until
all neighbours have acknowledged the packet, up to some number of
times. If the number of neighbours is large, however, requesting
acknowledgements from all of them might cause a non-negligible amount
of network traffic; in that case, it may be preferable to simply
repeat the update some reasonable number of times (say, 5 for
wireless and 2 for wired links).

A route retraction is somewhat less worrying: if the route retraction
doesn't reach all neighbours, a black-hole might be created, which,
unlike a routing loop, does not endanger the integrity of the
network. When a route is retracted, a node SHOULD send a triggered
update and SHOULD make a reasonable attempt at ensuring that all
neighbours receive this retraction.

Finally, a node MAY send a triggered update when the metric for a
given prefix changes in a significant manner, either due to a
received update or because a link cost has changed. A node SHOULD
NOT send triggered updates for other reasons, such as when there is a
minor fluctuation in a route's metric, when the selected next hop
changes, or to propagate a new sequence number (except to satisfy a
request, as specified in Section 3.8).

3.7.3. Maintaining Feasibility Distances

Before sending an update (prefix, plen, router-id, seqno, metric)
with finite metric (i.e., not a route retraction), a Babel node
updates the feasibility distance maintained in the source table.
This is done as follows.

If no entry indexed by (prefix, plen, router-id) exists in the source
table, then one is created with value (prefix, plen, router-id,
seqno, metric).

If an entry (prefix, plen, router-id, seqno', metric') exists, then
it is updated as follows:

o if seqno > seqno', then seqno' := seqno, metric' := metric;

o if seqno = seqno' and metric' > metric, then metric' := metric;

o otherwise, nothing needs to be done.

The garbage-collection timer for the modified entry is then reset.
Note that the garbage-collection timer is not reset when a retraction
is sent.

3.7.4. Split Horizon

When running over a transitive, symmetric link technology, e.g., a
point-to-point link or a wired LAN technology such as Ethernet, a
Babel node SHOULD use an optimisation known as split horizon. When
split horizon is used on a given interface, a routing update is not
sent on this particular interface when the advertised route was
learnt from a neighbour over the same interface.

Split horizon SHOULD NOT be applied to an interface unless the
interface is known to be symmetric and transitive; in particular,
split horizon is not applicable to decentralised wireless link
technologies (e.g., IEEE 802.11 in ad hoc mode).

3.8. Explicit Route Requests

In normal operation, a node's routing table is populated by the
regular and triggered updates sent by its neighbours. Under some
circumstances, however, a node sends explicit requests to cause a
resynchronisation with the source after a mobility event or to
prevent a route from spuriously expiring.

The Babel protocol provides two kinds of explicit requests: route
requests, which simply request an update for a given prefix, and
seqno requests, which request an update for a given prefix with a
specific sequence number. The former are never forwarded; the latter
are forwarded if they cannot be satisfied by a neighbour.

3.8.1. Handling Requests

Upon receiving a request, a node either forwards the request or sends
an update in reply to the request, as described in the following
sections. If this causes an update to be sent, the update is either
sent to a multicast address on the interface on which the request was
received, or to the unicast address of the neighbour that sent the
update.

The exact behaviour is different for route requests and seqno
requests.

3.8.1.1. Route Requests

   When a node receives a route request for a prefix (prefix, plen), it
   checks its route table for a selected route to this exact prefix. If
   such a route exists, it MUST send an update; if such a route does
   not, it MUST send a retraction for that prefix.

   When a node receives a wildcard route request, it SHOULD send a full
   routing table dump.

3.8.1.2. Seqno Requests

   When a node receives a seqno request for a given router-id and
   sequence number, it checks whether its routing table contains a
   selected entry for that prefix; if no such entry exists, or the entry
   has infinite metric, it ignores the request.

   If a selected route for the given prefix exists, and either the
   router-ids are different or the router-ids are equal and the entry's
   sequence number is no smaller than the requested sequence number, it
   MUST send an update for the given prefix.

   If the router-ids match but the requested seqno is larger than the
   route entry's, the node compares the router-id against its own
   router-id. If the router-id is its own, then it increases its
   sequence number by 1 and sends an update. A node MUST NOT increase
   its sequence number by more than 1 in response to a route request.

   If the requested router-id is not its own, the received request's hop
   count is 2 or more, and the node has a route (not necessarily a
   feasible one) for the requested prefix that does not use the
   requestor as a next hop, the node SHOULD forward the request. It
   does so by decreasing the hop count and sending the request in a
   unicast packet destined to a neighbour that advertises the given
   prefix (not necessarily the selected neighbour) and that is distinct
   from the neighbour from which the request was received.

   A node SHOULD maintain a list of recently forwarded requests and
   forward the reply in a timely manner. A node SHOULD compare every
   incoming request against its list of recently forwarded requests and
   avoid forwarding it if it is redundant.

   Since the request-forwarding mechanism does not necessarily obey the
   feasibility condition, it may get caught in routing loops; hence,
   requests carry a hop count to limit the time for which they remain in
   the network. However, since requests are only ever forwarded as
   unicast packets, the initial hop count need not be kept particularly

low, and performing an expanding horizon search is not necessary. A
request MUST NOT be forwarded to a multicast address, and it MUST be
forwarded to a single neighbour only.

3.8.2. Sending Requests

A Babel node MAY send a route or seqno request at any time, to a
multicast or a unicast address; there is only one case when
originating requests is required (Section 3.8.2.1).

3.8.2.1. Avoiding Starvation

When a route is retracted or expires, a Babel node usually switches
to another feasible route for the same prefix. It may be the case,
however, that no such routes are available.

A node that has lost all feasible routes to a given destination MUST
send a seqno request. The router-id of the request is set to the
router-id of the route that it has just lost, and the requested seqno
is the value contained in the source table, plus 1.

Such a request SHOULD be multicast over all of the node's attached
interfaces. Similar requests will be sent by other nodes that are
affected by the route's loss and will be forwarded by neighbouring
nodes up to the source. If the network is connected, and there is no
packet loss, this will result in a route being advertised with a new
sequence number. (Note that, due to duplicate suppression, only a
small number of such requests will actually reach the source.)

In order to compensate for packet loss, a node SHOULD repeat such a
request a small number of times if no route becomes feasible within a
short time. Under heavy packet loss, however, all such requests may
be lost; in that case, the second mechanism in the next section will
eventually ensure that a new seqno is received.

3.8.2.2. Dealing with Unfeasible Updates

When a route's metric increases, a node might receive an unfeasible
update for a route that it has currently selected. As specified in
Section 3.5.1, the receiving node will either ignore the update or
retract the route.

In order to keep routes from spuriously expiring because they have
become unfeasible, a node SHOULD send a unicast seqno request
whenever it receives an unfeasible update for a route that is
currently selected. The requested sequence number is computed from
the source table as above.

Additionally, a node SHOULD send a unicast seqno request whenever it
receives an unfeasible update from a currently unselected neighbour
that is "good enough", i.e., that would lead to the received route
becoming selected were it feasible.

3.8.2.3. Preventing Routes from Expiring

In normal operation, a route's expiry timer should never trigger:
since a route's hold time is computed from an explicit interval
included in Update TLVs, a new update should arrive in time to
prevent a route from expiring.

In the presence of packet loss, however, it may be the case that no
update is successfully received for an extended period of time,
causing a route to expire. In order to avoid such spurious expiry,
shortly before a selected route expires, a Babel node SHOULD send a
unicast route request to the neighbour that advertised this route;
since nodes always send retractions in response to non-wildcard route
requests (Section 3.8.1.1), this will usually result in either the
route being refreshed or a retraction being received.

3.8.2.4. Acquiring New Neighbours

In order to speed up convergence after a mobility event, a node MAY
send a unicast wildcard request after acquiring a new neighbour.
Additionally, a node MAY send a small number of multicast wildcard
requests shortly after booting.

4. Protocol Encoding

A Babel packet is sent as the body of a UDP datagram, with network-
layer hop count set to 1, destined to a well-known multicast address
or to a unicast address, over IPv4 or IPv6; in the case of IPv6,
these addresses are link-local. Both the source and destination UDP
port are set to a well-known port number. A Babel packet MUST be
silently ignored unless its source address is either a link-local
IPv6 address, or an IPv4 address belonging to the local network, and
its source port is the well-known Babel port. Babel packets MUST NOT
be sent as IPv6 Jumbograms.

In order to minimise the number of packets being sent while avoiding
lower-layer fragmentation, a Babel node SHOULD attempt to maximise
the size of the packets it sends, up to the outgoing interface's MTU
adjusted for lower-layer headers (28 octets for UDP/IPv4, 48 octets
for UDP/IPv6). It MUST NOT send packets larger than the attached
interface's MTU (adjusted for lower-layer headers) or 512 octets,
whichever is larger, but not exceeding $2^{16} - 1$ adjusted for lower-

layer headers. Every Babel speaker MUST be able to receive packets
that are as large as any attached interface's MTU (adjusted for
lower-layer headers) or 512 octets, whichever is larger.

In order to avoid global synchronisation of a Babel network and to
aggregate multiple TLVs into large packets, a Babel node MUST buffer
every TLV and delay sending a UDP packet by a small, randomly chosen
delay [JITTER]. In order to allow accurate computation of packet
loss rates, this delay MUST NOT be larger than half the advertised
Hello interval.

4.1. Data Types

4.1.1. Interval

   Relative times are carried as 16-bit values specifying a number of
   centiseconds (hundredths of a second). This allows times up to
   roughly 11 minutes with a granularity of 10 ms, which should cover
   all reasonable applications of Babel.

4.1.2. Router-Id

   A router-id is an arbitrary 8-octet value. Router-ids SHOULD be
   assigned in modified EUI-64 format [ADDRARCH].

4.1.3. Address

   Since the bulk of the protocol is taken by addresses, multiple ways
   of encoding addresses are defined. Additionally, a common subnet
   prefix may be omitted when multiple addresses are sent in a single
   packet -- this is known as address compression [PACKETBB].

   Address encodings:

   o AE 0: wildcard address. The value is 0 octets long.

   o AE 1: IPv4 address. Compression is allowed. 4 octets or less.

   o AE 2: IPv6 address. Compression is allowed. 16 octets or less.

   o AE 3: link-local IPv6 address. The value is 8 octets long, a
      prefix of fe80::/64 is implied.

   The address family of an address is either IPv4 or IPv6; it is
   undefined for AE 0, IPv4 for AE 1, and IPv6 for AE 2 and 3.

4.1.4. Prefixes

   A network prefix is encoded just like a network address, but it is
   stored in the smallest number of octets that are enough to hold the
   significant bits (up to the prefix length).

4.2. Packet Format

   A Babel packet consists of a 4-octet header, followed by a sequence
   of TLVs.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Magic     |    Version    |         Body length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Packet Body ...
+-+-+-+-+-+-+-+-+-+-+-+-
```
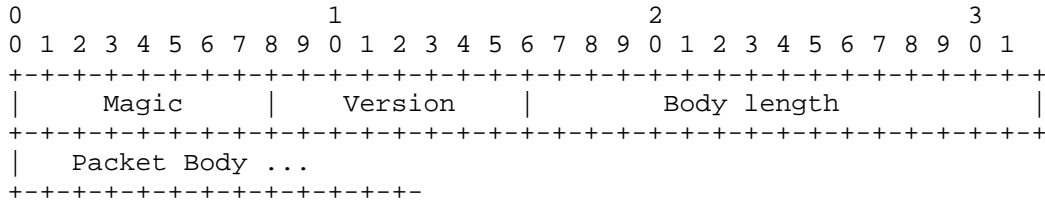
   Fields :

   Magic   The arbitrary but carefully chosen value 42 (decimal);
             packets with a first octet different from 42 MUST be
             silently ignored.

   Version   This document specifies version 2 of the Babel protocol.
             Packets with a second octet different from 2 MUST be
             silently ignored.

   Body length The length in octets of the body following the packet
                header.

   Body   The packet body; a sequence of TLVs.

   Any data following the body MUST be silently ignored.

4.3. TLV Format

   With the exception of Pad1, all TLVs have the following structure:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |    Body...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

   Fields :

   Type   The type of the TLV.

   Length   The length of the body, exclusive of the Type and Length
               fields. If the body is longer than the expected length of
               a given type of TLV, any extra data MUST be silently
               ignored.

   Body   The TLV body, the interpretation of which depends on the
               type.

   TLVs with an unknown type value MUST be silently ignored.

4.4. Details of Specific TLVs

4.4.1. Pad1

   0
   0 1 2 3 4 5 6 7
   +-+-+-+-+-+-+-+-+
   | Type = 0 |
   +-+-+-+-+-+-+-+-+

   Fields :

   Type   Set to 0 to indicate a Pad1 TLV.

   This TLV is silently ignored on reception.

4.4.2. PadN

   0   1   2   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+
   | Type = 1 | Length | MBZ...
   +-

   Fields :

   Type   Set to 1 to indicate a PadN TLV.

   Length   The length of the body, exclusive of the Type and Length
               fields.

   MBZ   Set to 0 on transmission.

   This TLV is silently ignored on reception.

4.4.3. Acknowledgement Request

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 2    |    Length     |            Reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Nonce              |            Interval           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This TLV requests that the receiver send an Acknowledgement TLV
within the number of centiseconds specified by the Interval field.

Fields :

Type   Set to 2 to indicate an Acknowledgement Request TLV.

Length   The length of the body, exclusive of the Type and Length
          fields.

Reserved Sent as 0 and MUST be ignored on reception.

Nonce   An arbitrary value that will be echoed in the receiver's
          Acknowledgement TLV.

Interval A time interval in centiseconds after which the sender will
          assume that this packet has been lost. This MUST NOT be 0.
          The receiver MUST send an acknowledgement before this time
          has elapsed (with a margin allowing for propagation time).

4.4.4. Acknowledgement

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |    Length     |            Nonce              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
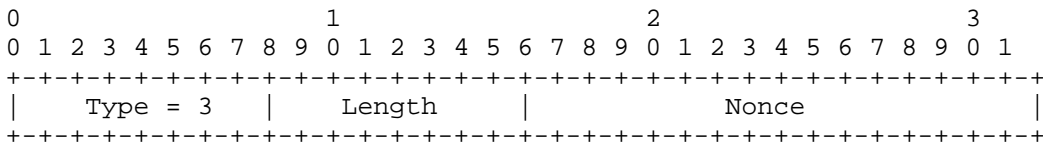
This TLV is sent by a node upon receiving an Acknowledgement Request.

Fields :

Type   Set to 3 to indicate an Acknowledgement TLV.

Length   The length of the body, exclusive of the Type and Length
          fields.

   Nonce   Set to the Nonce value of the Acknowledgement Request that
              prompted this Acknowledgement.

   Since nonce values are not globally unique, this TLV MUST be sent to
   a unicast address.

4.4.5. Hello

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 4    |    Length     |             Reserved          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Seqno             |            Interval           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   This TLV is used for neighbour discovery and for determining a link's
   reception cost.

   Fields :

   Type   Set to 4 to indicate a Hello TLV.

   Length   The length of the body, exclusive of the Type and Length
              fields.

   Reserved   Sent as 0 and MUST be ignored on reception.

   Seqno   The value of the sending node's Hello seqno for this
              interface.

   Interval   An upper bound, expressed in centiseconds, on the time
              after which the sending node will send a new Hello TLV.
              This MUST NOT be 0.

   Since there is a single seqno counter for all the Hellos sent by a
   given node over a given interface, this TLV MUST be sent to a
   multicast destination. In order to avoid large discontinuities in
   link quality, multiple Hello TLVs SHOULD NOT be sent in the same
   packet.

4.4.6. IHU

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 5    |    Length     |      AE       |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Rxcost             |           Interval            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Address...
+-+-+-+-+-+-+-+-+-+-+-
```
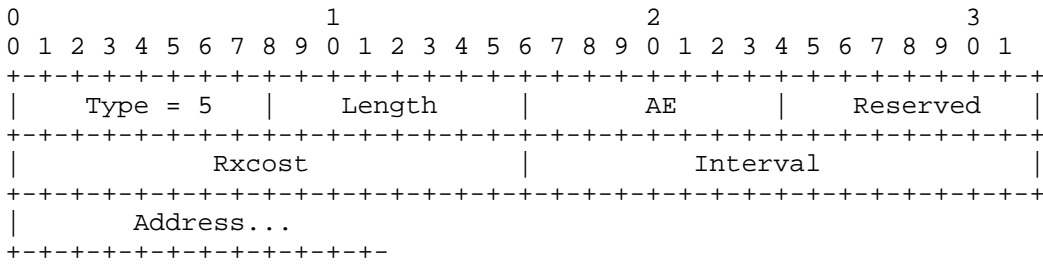
An IHU ("I Heard You") TLV is used for confirming bidirectional
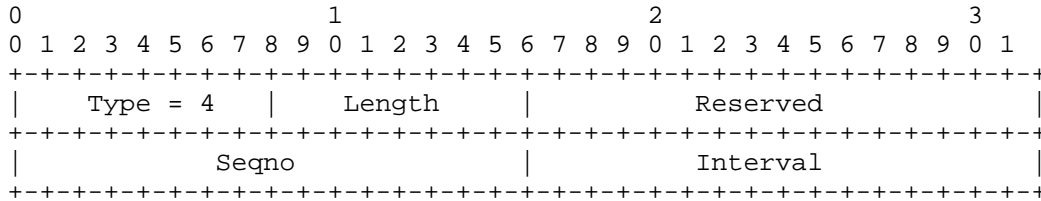reachability and carrying a link's transmission cost.

Fields :

Type   Set to 5 to indicate an IHU TLV.

Length   The length of the body, exclusive of the Type and Length
           fields.

AE   The encoding of the Address field. This should be 1 or 3
           in most cases. As an optimisation, it MAY be 0 if the TLV
           is sent to a unicast address, if the association is over a
           point-to-point link, or when bidirectional reachability is
           ascertained by means outside of the Babel protocol.

Reserved   Sent as 0 and MUST be ignored on reception.

Rxcost   The rxcost according to the sending node of the interface
           whose address is specified in the Address field. The value
           FFFF hexadecimal (infinity) indicates that this interface
           is unreachable.

Interval   An upper bound, expressed in centiseconds, on the time
           after which the sending node will send a new IHU; this MUST
           NOT be 0. The receiving node will use this value in order
           to compute a hold time for this symmetric association.

Address   The address of the destination node, in the format
           specified by the AE field. Address compression is not
           allowed.

Conceptually, an IHU is destined to a single neighbour. However, IHU
TLVs contain an explicit destination address, and it SHOULD be sent
to a multicast address, as this allows aggregation of IHUs destined

to distinct neighbours into a single packet and avoids the need for
an ARP or Neighbour Discovery exchange when a neighbour is not being
used for data traffic.

IHU TLVs with an unknown value for the AE field MUST be silently
ignored.

4.4.7. Router-Id

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 6    |    Length     |             Reserved          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                            Router-Id                          +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A Router-Id TLV establishes a router-id that is implied by subsequent
Update TLVs.

Fields :

Type   Set to 6 to indicate a Router-Id TLV.

Length   The length of the body, exclusive of the Type and Length
            fields.

Reserved   Sent as 0 and MUST be ignored on reception.

Router-Id   The router-id for routes advertised in subsequent Update
            TLVs

4.4.8. Next Hop

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 7    |    Length     |      AE       |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Next hop...
+-+-+-+-+-+-+-+-+-+-+-
```
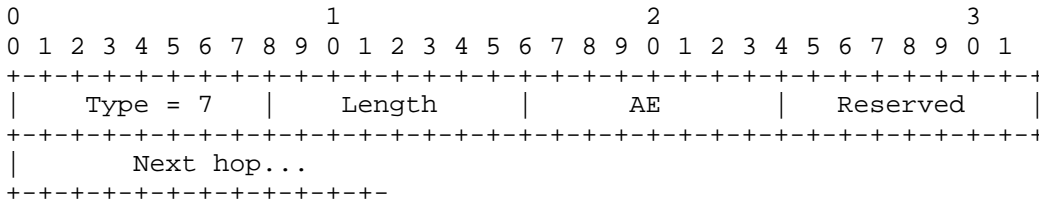
A Next Hop TLV establishes a next-hop address for a given address
family (IPv4 or IPv6) that is implied by subsequent Update TLVs.
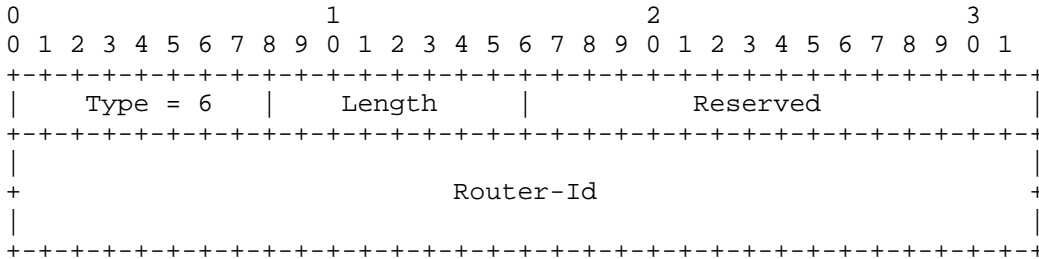
   Fields :

   Type   Set to 7 to indicate a Next Hop TLV.

   Length   The length of the body, exclusive of the Type and Length
               fields.

   AE   The encoding of the Address field. This SHOULD be 1 or 3
               and MUST NOT be 0.

   Reserved   Sent as 0 and MUST be ignored on reception.

   Next hop   The next-hop address advertised by subsequent Update TLVs,
               for this address family.

   When the address family matches the network-layer protocol that this
   packet is transported over, a Next Hop TLV is not needed: in that
   case, the next hop is taken to be the source address of the packet.

   Next Hop TLVs with an unknown value for the AE field MUST be silently
   ignored.

4.4.9. Update

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = 8   |    Length     |      AE       |     Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Plen      |    Omitted    |            Interval           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Seqno              |            Metric             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Prefix...
+-+-+-+-+-+-+-+-+-+-+-+-
```

   An Update TLV advertises or retracts a route. As an optimisation,
   this can also have the side effect of establishing a new implied
   router-id and a new default prefix.

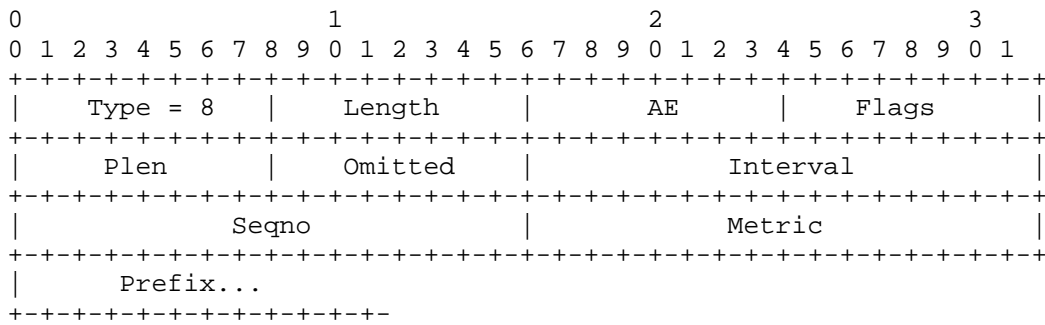   Fields :

   Type   Set to 8 to indicate an Update TLV.

   Length   The length of the body, exclusive of the Type and Length
               fields.

   AE   The encoding of the Prefix field.

Flags   The individual bits of this field specify special handling
           of this TLV (see below). Every node MUST be able to
           interpret the flags with values 80 and 40 hexadecimal;
           unknown flags MUST be silently ignored.

Plen   The length of the advertised prefix.

Omitted   The number of octets that have been omitted at the
           beginning of the advertised prefix and that should be taken
           from a preceding Update TLV with the flag with value 80
           hexadecimal set.

Interval   An upper bound, expressed in centiseconds, on the time
           after which the sending node will send a new update for
           this prefix. This MUST NOT be 0 and SHOULD NOT be less
           than 10. The receiving node will use this value to compute
           a hold time for this routing table entry. The value FFFF
           hexadecimal (infinity) expresses that this announcement
           will not be repeated unless a request is received
           (Section 3.8.2.3).

Seqno   The originator's sequence number for this update.

Metric   The sender's metric for this route. The value FFFF
           hexadecimal (infinity) means that this is a route
           retraction.

Prefix   The prefix being advertised. This field's size is (Plen/8
           - Omitted) rounded upwards.

The Flags field is interpreted as follows:

o   if the bit with value 80 hexadecimal is set, then this Update
    establishes a new default prefix for subsequent Update TLVs with a
    matching address family within the same packet;

o   if the bit with value 40 hexadecimal is set, then the low-order 8
    octets of the advertised prefix establish a new default router-id
    for this TLV and subsequent Update TLVs in the same packet.

The prefix being advertised by an Update TLV is computed as follows:

o   the first Omitted octets of the prefix are taken from the previous
    Update TLV with flag 80 hexadecimal set and the same address
    family;

o   the next (Plen/8 - Omitted) (rounded upwards) octets are taken
    from the Prefix field;
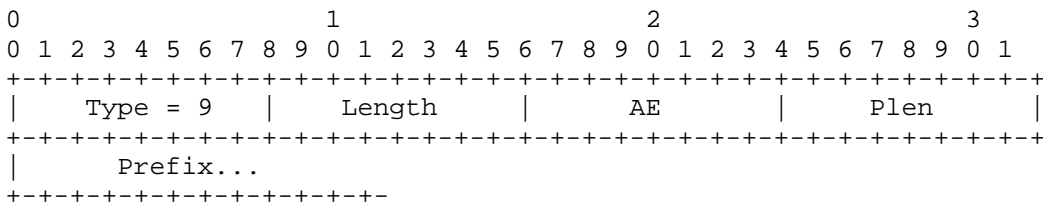
o the remaining octets are set to 0.

If the Metric field is finite, the router-id of the originating node
for this announcement is taken from the low-order 8 octets of the
prefix advertised by this Update if the bit with value 40 hexadecimal
is set in the Flags field. Otherwise, it is taken either from the
preceding Router-Id packet, or the preceding Update packet with flag
40 hexadecimal set, whichever comes last.

The next-hop address for this update is taken from the last preceding
Next Hop TLV with a matching address family in the same packet; if no
such TLV exists, it is taken from the network-layer source address of
this packet.

If the metric field is FFFF hexadecimal, this TLV specifies a
retraction. In that case, the current router-id and the Seqno are
not used. AE MAY then be 0, in which case this Update retracts all
of the routes previously advertised on this interface.

Update TLVs with an unknown value for the AE field MUST be silently
ignored.

4.4.10. Route Request

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = 9   |    Length     |      AE       |      Plen     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Prefix...
+-+-+-+-+-+-+-+-+-+-+-
```

A Route Request TLV prompts the receiver to send an update for a
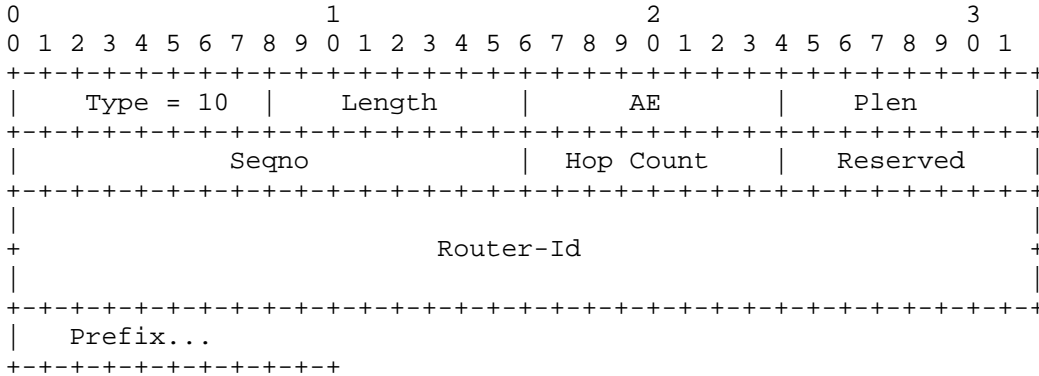given prefix, or a full routing table dump.

Fields :

Type   Set to 9 to indicate a Route Request TLV.

Length   The length of the body, exclusive of the Type and Length
          fields.

AE   The encoding of the Prefix field. The value 0 specifies
          that this is a request for a full routing table dump (a
          wildcard request).

Plen   The length of the requested prefix.

   Prefix   The prefix being requested. This field's size is Plen/8
              rounded upwards.

   A Request TLV prompts the receiving node to send an update message
   for the prefix specified by the AE, Plen, and Prefix fields, or a
   full dump of its routing table if AE is 0 (in which case Plen MUST be
   0, and Prefix is of length 0). A Request may be sent to a unicast
   address if it is destined to a single node, or to a multicast address
   if the request is destined to all of the neighbours of the sending
   interface.

4.4.11. Seqno Request

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = 10  |     Length    |       AE      |      Plen     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Seqno            |   Hop Count   |    Reserved   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                           Router-Id                           +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Prefix...
+-+-+-+-+-+-+-+-+
```

   A Seqno Request TLV prompts the receiver to send an Update for a
   given prefix with a given sequence number, or to forward the request
   further if it cannot be satisfied locally.

   Fields :

   Type   Set to 10 to indicate a Seqno Request message.

   Length   The length of the body, exclusive of the Type and Length
              fields.

   AE   The encoding of the Prefix field. This MUST NOT be 0.

   Plen   The length of the requested prefix.

   Seqno   The sequence number that is being requested.

   Hop Count The maximum number of times that this TLV may be forwarded,
              plus 1. This MUST NOT be 0.

   Prefix   The prefix being requested. This field's size is Plen/8
             rounded upwards.

   A Seqno Request TLV prompts the receiving node to send an Update for
   the prefix specified by the AE, Plen, and Prefix fields, with either
   a router-id different from what is specified by the Router-Id field,
   or a Seqno no less than what is specified by the Seqno field. If
   this request cannot be satisfied locally, then it is forwarded
   according to the rules set out in Section 3.8.1.2.

   While a Seqno Request MAY be sent to a multicast address, it MUST NOT
   be forwarded to a multicast address and MUST NOT be forwarded to more
   than one neighbour. A request MUST NOT be forwarded if its Hop Count
   field is 1.

5. IANA Considerations

   IANA has registered the UDP port number 6697, called "babel", for use
   by the Babel protocol.

   IANA has registered the IPv6 multicast group ff02:0:0:0:0:0:1:6 and
   the IPv4 multicast group 224.0.0.111 for use by the Babel protocol.

6. Security Considerations

   As defined in this document, Babel is a completely insecure protocol.
   Any attacker can attract data traffic by advertising routes with a
   low metric. This particular issue can be solved either by lower-
   layer security mechanisms (e.g., IPsec or link-layer security), or by
   appending a cryptographic key to Babel packets; the provision of
   ignoring any data contained within a Babel packet beyond the body
   length declared by the header is designed for just such a purpose.

   The information that a Babel node announces to the whole routing
   domain is often sufficient to determine a mobile node's physical
   location with reasonable precision. The privacy issues that this
   causes can be mitigated somewhat by using randomly chosen router-ids
   and randomly chosen IP addresses, and changing them periodically.

   When carried over IPv6, Babel packets are ignored unless they are
   sent from a link-local IPv6 address; since routers don't forward
   link-local IPv6 packets, this provides protection against spoofed
   Babel packets being sent from the global Internet. No such natural
   protection exists when Babel packets are carried over IPv4.

7. References

7.1. Normative References

   [ADDRARCH] Hinden, R. and S. Deering, "IP Version 6 Addressing
               Architecture", RFC 4291, February 2006.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

   [DSDV]   Perkins, C. and P. Bhagwat, "Highly Dynamic Destination-
               Sequenced Distance-Vector Routing (DSDV) for Mobile
               Computers", ACM SIGCOMM'94 Conference on Communications
               Architectures, Protocols and Applications 234-244, 1994.

   [DUAL]   Garcia Luna Aceves, J., "Loop-Free Routing Using
               Diffusing Computations", IEEE/ACM Transactions on
               Networking 1:1, February 1993.

   [EIGRP]   Albrightson, B., Garcia Luna Aceves, J., and J. Boyle,
               "EIGRP -- a Fast Routing Protocol Based on Distance
               Vectors", Proc. Interop 94, 1994.

   [ETX]   De Couto, D., Aguayo, D., Bicket, J., and R. Morris, "A
               high-throughput path metric for multi-hop wireless
               networks", Proc. MobiCom 2003, 2003.

   [IS-IS]   "Information technology -- Telecommunications and
               information exchange between systems -- Intermediate
               System to Intermediate System intra-domain routeing
               information exchange protocol for use in conjunction with
               the protocol for providing the connectionless-mode
               network service (ISO 8473)", ISO/IEC 10589:2002.

   [JITTER]   Floyd, S. and V. Jacobson, "The synchronization of
               periodic routing messages", IEEE/ACM Transactions on
               Networking 2, 2, 122-136, April 1994.

   [OSPF]   Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

   [PACKETBB] Clausen, T., Dearlove, C., Dean, J., and C. Adjih,
               "Generalized Mobile Ad Hoc Network (MANET) Packet/Message
               Format", RFC 5444, February 2009.

   [RIP]   Malkin, G., "RIP Version 2", STD 56, RFC 2453,
               November 1998.

Appendix A. Cost and Metric Computation

   The strategy for computing link costs and route metrics is a local
   matter; Babel itself only requires that it comply with the conditions
   given in Sections 3.4.3 and 3.5.2. Different nodes MAY use different
   strategies in a single network and MAY use different strategies on
   different interface types. This section gives a few examples of such
   strategies.

   The sample implementation of Babel maintains statistics about the
   last 16 received Hello TLVs (Appendix A.1), computes costs by using
   the 2-out-of-3 strategy (Appendix A.2.1) on wired links, and ETX
   [ETX] on wireless links. It uses an additive algebra for metric
   computation (Appendix A.3.1).

A.1. Maintaining Hello History

   For each neighbour, the sample implementation of Babel maintains a
   Hello history and an expected sequence number. The Hello history is
   a vector of 16 bits, where a 1 value represents a received Hello, and
   a 0 value a missed Hello. The expected sequence number, written ne,
   is the sequence number that is expected to be carried by the next
   received hello from this neighbour.

   Whenever it receives a Hello packet from a neighbour, a node compares
   the received sequence number nr with its expected sequence number ne.
   Depending on the outcome of this comparison, one of the following
   actions is taken:

   o if the two differ by more than 16 (modulo 2^{16}), then the sending
      node has probably rebooted and lost its sequence number; the
      associated neighbour table entry is flushed;

   o otherwise, if the received nr is smaller (modulo 2^{16}) than the
      expected sequence number ne, then the sending node has increased
      its Hello interval without our noticing; the receiving node
      removes the last (ne - nr) entries from this neighbour's Hello
      history (we "undo history");

   o otherwise, if nr is larger (modulo 2^{16}) than ne, then the sending
      node has decreased its Hello interval, and some Hellos were lost;
      the receiving node adds (nr - ne) 0 bits to the Hello history (we
      "fast-forward").

The receiving node then appends a 1 bit to the neighbour's Hello
history, resets the neighbour's Hello timer, and sets ne to (nr + 1).
It then resets the neighbour's Hello timer to 1.5 times the value
advertised in the received Hello (the extra margin allows for the
delay due to jitter).

Whenever the Hello timer associated to a neighbour expires, the local
node adds a 0 bit to this neighbour's Hello history, and increments
the expected Hello number. If the Hello history is empty (it
contains 0 bits only), the neighbour entry is flushed; otherwise, it
resets the neighbour's Hello timer to the value advertised in the
last Hello received from this neighbour (no extra margin is necessary
in this case).

A.2. Cost Computation

A.2.1. k-out-of-j

   K-out-of-j link sensing is suitable for wired links that are either
   up, in which case they only occasionally drop a packet, or down, in
   which case they drop all packets.

   The k-out-of-j strategy is parameterised by two small integers k and
   j, such that $0 < k <= j$, and the nominal link cost, a constant $K >=$
   1. A node keeps a history of the last j hellos; if k or more of
   those have been correctly received, the link is assumed to be up, and
   the rxcost is set to K; otherwise, the link is assumed to be down,
   and the rxcost is set to infinity.

   The cost of such a link is defined as

   o   cost = FFFF hexadecimal if rxcost = FFFF hexadecimal;

   o   cost = txcost otherwise.

A.2.2. ETX

   The Estimated Transmission Cost metric [ETX] estimates the number of
   times that a unicast frame will be retransmitted by the IEEE 802.11
   MAC, assuming infinite persistence.

   A node uses a neighbour's Hello history to compute an estimate,
   written beta, of the probability that a Hello TLV is successfully
   received. The rxcost is defined as 256/beta.

   Let alpha be MIN(1, 256/txcost), an estimate of the probability of
   successfully sending a Hello TLV. The cost is then computed by

```
      cost = 256/(alpha * beta)
```

   or, equivalently,

```
      cost = (MAX(txcost, 256) * rxcost) / 256.
```

A.3. Metric Computation

A.3.1. Additive Metrics

   The simplest approach for obtaining a monotonic, isotonic metric is
   to define the metric of a route as the sum of the costs of the
   component links. More formally, if a neighbour advertises a route
   with metric m over a link with cost c, then the resulting route has
   metric $M(c, m) = c + m$.

   A multiplicative metric can be converted to an additive one by taking
   the logarithm (in some suitable base) of the link costs.

A.3.2. External Sources of Willingness

   A node may want to vary its willingness to forward packets by taking
   into account information that is external to the Babel protocol, such
   as the monetary cost of a link, the node's battery status, CPU load,
   etc. This can be done by adding to every route's metric a value k
   that depends on the external data. For example, if a battery-powered
   node receives an update with metric m over a link with cost c, it
   might compute a metric $M(c, m) = k + c + m$, where k depends on the
   battery status.

   In order to preserve strict monotonicity (Section 3.5.2), the value k
   must be greater than -c.

Appendix B. Constants

   The choice of time constants is a trade-off between fast detection of
   mobility events and protocol overhead. Two implementations of Babel
   with different time constants will interoperate, although the
   resulting convergence time will most likely be dictated by the
   slowest of the two implementations.

   Experience with the sample implementation of Babel indicates that the
   Hello interval is the most important time constant: a mobility event
   is detected within 1.5 to 3 Hello intervals. Due to Babel's reliance
   on triggered updates and explicit requests, the Update interval only
   has an effect on the time it takes for accurate metrics to be
   propagated after variations in link costs too small to trigger an
   unscheduled update.

At the time of writing, the sample implementation of Babel uses the
following default values:

    Hello Interval: 4 seconds on wireless links, 20 seconds on wired
    links.

    IHU Interval: the advertised IHU interval is always 3 times the
    Hello interval. IHUs are actually sent with each Hello on lossy
    links (as determined from the Hello history), but only with every
    third Hello on lossless links.

    Update Interval: 4 times the Hello interval.

    IHU Hold Time: 3.5 times the advertised IHU interval.

    Route Expiry Time: 3.5 times the advertised update interval.

    Source GC time: 3 minutes.

The amount of jitter applied to a packet depends on whether it
contains any urgent TLVs or not. Urgent triggered updates and urgent
requests are delayed by no more than 200 ms; other TLVs are delayed
by no more than one-half the Hello interval.

Appendix C. Simplified Implementations

Babel is a fairly economic protocol. Route updates take between 12
and 40 octets per destination, depending on how successful
compression is; in a double-stack mesh network, an average of less
than 24 octets is typical. The route table occupies about 35 octets
per IPv6 entry. To put these values into perspective, a single full-
size Ethernet frame can carry some 65 route updates, and a megabyte
of memory can contain a 20000-entry routing table and the associated
source table.

Babel is also a reasonably simple protocol. The sample
implementation consists of less than 8000 lines of C code, and it
compiles to less than 60 kB of text on a 32-bit CISC architecture.

Nonetheless, in some very constrained environments, such as PDAs,
microwave ovens, or abacuses, it may be desirable to have subset
implementations of the protocol.

A parasitic implementation is one that uses a Babel network for
routing its packets but does not announce any of the routes that it
has learnt from its neighbours. (This is slightly more than a
passive implementation, which doesn't even announce routes to
itself.) It may either maintain a full routing table or simply

select a gateway amongst any one of its neighbours that announces a
default route. Since a parasitic implementation never forwards
packets, it cannot possibly participate in a routing loop; hence, it
need not evaluate the feasibility condition and need not maintain a
source table.

A parasitic implementation MUST answer acknowledgement requests and
MUST participate in the Hello/IHU protocol. Finally, it MUST be able
to reply to seqno requests for routes that it announces and SHOULD be
able to reply to route requests.

Appendix D. Software Availability

The sample implementation of Babel is available from
<http://www.pps.jussieu.fr/~jch/software/babel/>.

Author's Address

Juliusz Chroboczek
PPS, University of Paris 7
Case 7014
75205 Paris Cedex 13,
France

EMail: jch@pps.jussieu.fr