

# HLP Implementation

## Version 0.1

International Computer Science Institute

## 1 Introduction

Hybrid Link-State Path-Vector Protocol, or HLP, is an inter-domain routing protocol designed as a replacement for the current Border Gateway Protocol (BGP). Using a combination of link-state and path vector routing, it provides greater scalability, better fault isolation and better convergence.

The core of HLP is the inclusion of the economic and political structure of the Internet into inter-domain routing. That is, BGP currently considers each AS as a node in a general graph without any specific structure (using explicit policies to constrain routing), whereas HLP assumes that the Internet structure is basically hierarchical with the provider autonomous systems (ASs) being the roots of customer ASs.

HLP explicitly includes the relationship between 2 neighboring ASs in its protocol. This will reduce misconfigurations which should hopefully reduce the occurrence of routing abnormalities. However, the tradeoff is some amount of inflexibility in the routing algorithm. This is resolved using exceptions that are expected to be rare and therefore acceptable.

This report summarizes implementation of HLP on the XORP[1] software router. The implementation reuses much of the code in XORP's BGP module.

## 2 Definitions

We say that two ASs are in the same *hierarchy* if there exists a directed path between them such that the path consists of any number of provider links followed by any number of customer links. This definition of hierarchy implies that there exists at least one route between two ASs in a hierarchy that does not include (provider)  $\rightarrow$  (customer)  $\rightarrow$  (provider) links. Two ASs are *neighbors* if there exists a link between them. The relationship between neighboring ASs (peer, customer, or provider) determine the overall structure of the network, which can be as simple as shown in Figure 1a, or consist of overlapping hierarchies as shown in Figure 1b. In the figure, each node represents an AS; the neighboring AS at a higher tier is the provider, similarly an AS at a lower tier is the customer. Thus, AS A is the provider of AS B, which in turn is A's customer. Neighboring ASs at the same tier are also called peering ASs. Note that the use of tiers in Figure 1 and in subsequent figures only allows for graphical representation of relationships, it is not present in the actual HLP protocol.

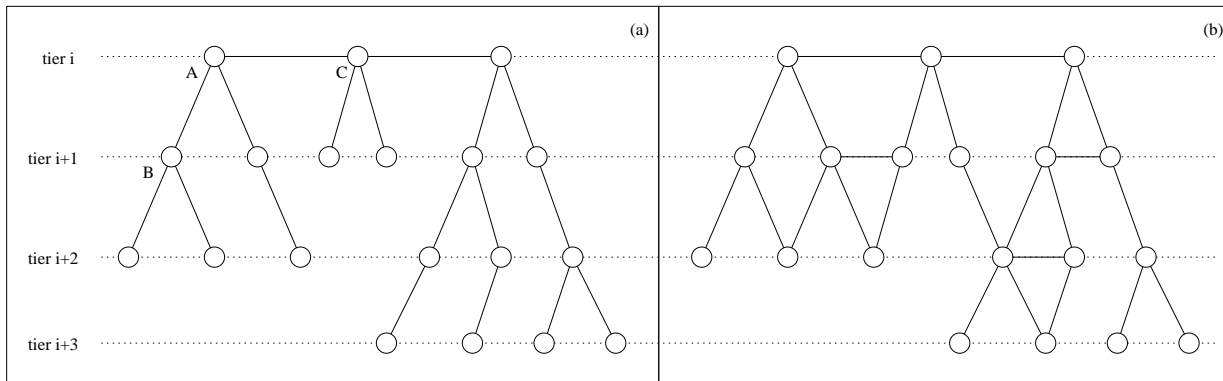


Figure 1: (a) Simple and (b) complex AS hierarchies

HLP divides the network into hierarchies consisting of providers and their customer ASs, and peering ASs in different hierarchies allow routing between hierarchies. As will be explained in the next section, this division of the network into

separate components increases scalability, as well as reduces the convergence time for route updates.

### 3 Routing Information Dissemination

Two types of routing packets are used to disseminate routing information: link-state advertisements (LSAs) and fragmented path-vector (FPV)<sup>1</sup>. As is the case in OSPF, LSAs are flooded throughout a hierarchy, and allow construction of the entire hierarchy topology. FPVs are used to route between ASs; they contain the numbers of peering ASs between hierarchies. LSAs that have not previously been received are forwarded in the following manner:

1. if they are from customers, we forward to all neighbors, except the customers from which they arrive.
2. if they are from providers, we forward only to neighboring customers, not providers.

The objective of the above rules is to restrict LSAs to the hierarchies from which they originate. Failure to do so will imply that multi-homing ASs belonging to different hierarchies can cause LSAs to be disseminated throughout the entire Internet. The rules above implements this restriction, illustrated in Figure 2. By definition, AS C belongs to both hierarchies 1 and 2, and rule 1 allows LSAs from AS B to be forwarded to C. At C, rule 2 prevents LSAs from A from being forwarded to AS D. On the other hand, LSAs involving C will be disseminated in both hierarchies, which is correct since C is a member of both.

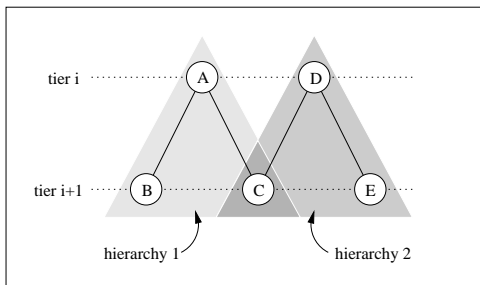


Figure 2: Example illustrating restriction of LSAs within originating hierarchies

FPVs are used to forward routes from one hierarchy to another. They are similar to path-vector packets used in BGP, except that they do not include the AS path within hierarchies. Rules that govern forwarding of FPVs are given as follows:

1. FPVs from providers are disseminated only to customers, not to peers or other providers.
2. FPVs from peers are forwarded to neighboring peers and customers, but not providers.

### 4 Routing Algorithm

The mechanism to choose a route to a particular destination AS is similar to that currently used in BGP. This eases the implementation of exceptions which will be covered in the next section, as well as the creation of FPVs for routes to customers within the hierarchy. Basically, we store routes for destination ASs reachable from each neighboring AS, then decide the winning route for a particular destination. The handling of routes contained within FPVs is straightforward and similar to that of BGP, but routing information gained from LSAs needs to be converted to the same form as that in FPVs. The conversion is elaborated on in the next section.

#### 4.1 From LSAs to Routes

We denote the least cost of reaching AS A from B by  $\text{cost}(B, A)$ , a route from A to B with cost C by  $\text{route}[(A, B), C]$ , and we perform the conversion in the following manner, assuming that the operations are taking place in AS X:

<sup>1</sup>FPVs contain only a portion of the AS path to the destination.

1. for each neighbor N
2.     if neighbor is a customer
3.         for each downstream customer AS A
4.             compute cost(N,A)
5.             create AS path [X,A] with cost C = [cost(N,A)+cost(X,N)]
6.             associate route[(X,A),C] with N
7.     else if neighbor is a provider
8.         for each of the non-customer ASs in the hierarchy
9.             compute cost(N,A)
10.             create AS path [X,A] with cost C = [cost(N,A)+cost(X,N)]
11.             associate route[(X,A),C] with N

The computation is broken into two parts, steps 2 to 5, and 6 to 9 of the algorithm above. This is required because forwarding of routes from a provider to a peer should take place only if the destination AS is a customer<sup>2</sup>.

To distinguish between the origin of the routes, we tag them with the following:

- PROVIDER\_FPV: Route contained within FPV sent from provider.
- PROVIDER\_LSA: Route for non-customer destination AS within the same hierarchy, determined using link-state information.
- PEER\_FPV: Route contained within FPV sent from peer.
- CUSTOMER\_LSA: Route for customer destination AS obtained using link-state information.

An example is given in Figure 3, where we focus on AS C. Table 1 shows the routes, costs and tags associated with each neighbor.

Neighbor	AS Path	Cost	Tag
A	(C,A)	2	PROVIDER_LSA
A	(C,B)	3	PROVIDER_LSA
A	(C,E)	6	PROVIDER_LSA
A	(C,D)	15	PROVIDER_LSA
D	(C,D)	17	PROVIDER_LSA
D	(C,E)	26	PROVIDER_LSA
D	(C,A)	32	PROVIDER_LSA
F	(C,F)	5	CUSTOMER_LSA

Table 1: Table containing routes, costs and tags for example converting LSAs to routes

## 4.2 From FPV to Routes

Creation of routes from FPVs are much simpler, and is similar to that in BGP:

1. if FPV is from a provider P
2.     extract route and metric, tag with PROVIDER\_FPV, and associate them with P
3. if FPV is from a peer Q
4.     extract route (prepending this AS' number) and metric, tag with PEER\_FPV, and associate with Q
5. if FPV is from customer, treat FPV as coming from peer, extract route (prepending this AS' number) and metric, tag with PEER\_FPV, and associate with Q

Note that step 5 only occurs due to an exception in the customer AS.

<sup>2</sup>An AS is a customer if there exists at least one route from the provider to that AS consisting only of provider-customer links.

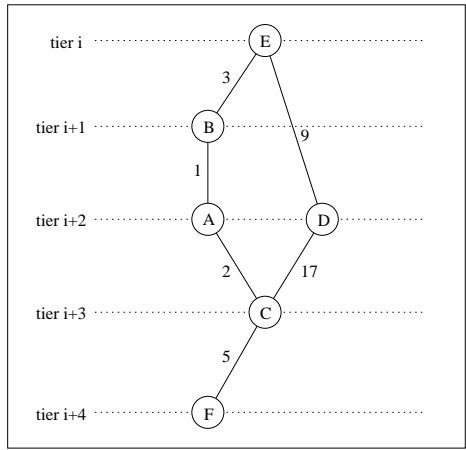


Figure 3: Example illustrating LSA to route conversion

### 4.3 Route Selection

The winning route to a particular destination is selected according to the following order of preferences:

1. customer route (ie. tagged with CUSTOMER\_LSA)
2. least metric (or cost)
3. shortest path length (number of peering ASs traversed)
4. neighboring AS with smallest ID
5. neighboring link with smallest address

Any exceptions to the above-mentioned rules will have to be configured by the operator. The types of exceptions supported are given in the next section.

### 4.4 Route Forwarding

The winning route may require forwarding to neighboring ASs. Note that in this section we consider only the forwarding of FPVs, since the choice of route to use is not disseminated in LSAs. To implement the rules given in Section 3, we forward a winning route to a neighboring AS only if that neighbor and the route tag is given in Table 2.

Route Tag	Neighbor Type
PROVIDER_FPV	Customer
PEER_FPV	Peer, Customer
CUSTOMER_LSA	Peer

Table 2: Route types that can be forwarded to corresponding neighboring AS types

## 5 Exceptions

HLP supports three different types of exceptions. The primary use of exceptions is to support operations that are currently used in BGP but not covered by the default HLP rules. The format in which exceptions are specified and stored is given by

$$\langle from\_link, to\_link, AS\ number \rangle$$

where

- *from\_link* specifies the neighboring link from which winning routes will be propagated,
- *to\_link* specifies the neighboring link to which winning routes will be propagated, and
- *AS number* refers to the destination AS that this particular exception is for.

In HLP, a *from\_link* or *to\_link* is given by the tuple:

$$\langle \text{this IP, this port, neighbor IP, neighbor port} \rangle$$

## 5.1 Exception 1

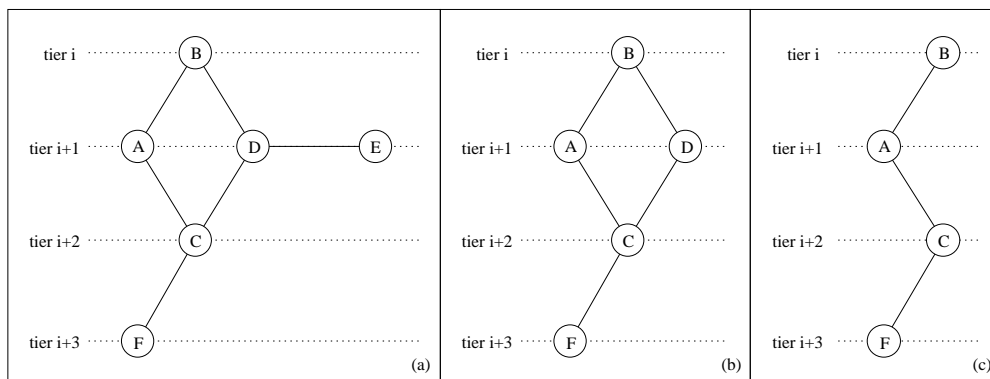


Figure 4: (a) Example illustrating effect of exception 1 on rest of hierarchy. D announces that it does not have a customer route to C. B uses graph in (b) to compute shortest paths to all ASs in the same hierarchy except C (i.e. A, D and F). B uses the graph in (c) to compute the shortest path to C.

The first exception, illustrated in Figure 4, allows an AS (say D) to choose an alternate route to a customer (C) via a peer (E). The route chosen is dependent on the customer AS, and should not affect routes to other ASs. D indicates its intention via LSAs to other ASs in the same hierarchy that it is not choosing customer routes to C. D also informs E that it no longer has a customer route to C. If, ignoring customer routes, the winning route is from E, then D forwards that FPV to its peers and customers. However, if the winning route is from another peer not specified in the exception, then the corresponding FPV will not be forwarded.

Using the same example, exception 1 is specified by

$$\langle E \rightarrow D, \text{NULL}, C \rangle$$

where *NULL* is specified using the tuple

$$\langle 0.0.0.0, 0, 0.0.0.0, 0 \rangle$$

## 5.2 Exception 2

This exception specifies that winning routes from the stated provider is to be forwarded to a particular peer, which is typically not done. In Figure 5, the exception

$$\langle A \rightarrow C, C \rightarrow D, Z \rangle$$

allows winning routes to AS Z from provider A to be forwarded to D.

## 5.3 Exception 3

The last exception supported is similar to exception 2. Here, routes are forwarded from a peer to a provider. Currently, the provider simply accepts incoming FPVs from customers, treating them as though they are from peers. If, for security reasons, providers should reject FPVs from customers, then additional configuration will be required in the provider.

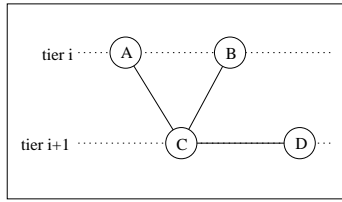


Figure 5: Simple network illustrating route forwarding from provider to peer

## 5.4 Exception Format and Matching

The type of exception does not explicitly need to be specified. Instead, the neighbor relationship associated with the links given in each exception rule can be used to determine this. The format is thus simplified, and should hopefully reduce the occurrence of misconfigurations. Table 3 gives the combination of links that indicate the kind of exception specified.

From_link	To_link	Exception Type
NULL	Peer	1
Provider	Peer	2
Peer	Provider	3

Table 3: Combination of link types associated with each exception type

## 6 Data Structures

In this section we describe the data structures used to maintain state in a HLP router. Figure 6 shows the flow of routing information through the system, as well as the major components of the system.

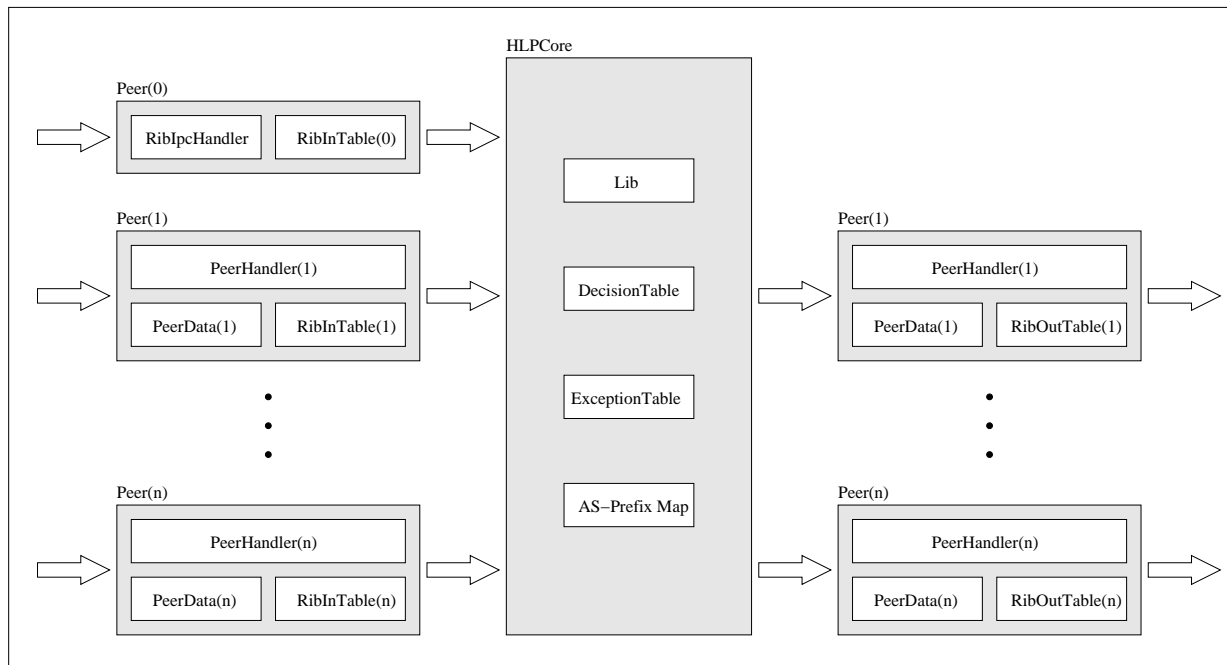


Figure 6: Flow of routing information through system

We describe each component below:

- *Peer*: A *peer* contains the necessary objects needed for receiving and sending routing information from and to neighboring routers. Each peer maintains the state machine for the connection associated with the corresponding neighbor, as well as the various timers needed during connection establishment and for keepalive messages.
- *RibIpcHandler*: Handles insertion of prefixes owned by this AS.
- *Routing Information Base Input Table (RibInTable)*: Stores routes associated with corresponding neighboring router. Routes may be obtained via FPVs sent from neighbor, or from computation of shortest paths using link-state information. The type of tag assigned to a route is explained in Section 4.1.
- *PeerHandler*: Handles reception and transmission of FPVs and LSAs. After a packet has been received, it is broken up into individual components (for instance, link changes, route withdrawals, route announcements, etc.) before being passed to HLPCore for processing. Updates passed from the DecisionTable are aggregated as much as possible within a packet (FPV or LSA) before being transmitted.
- *Routing Information Base Output Table (RibOutTable)*: Stores the outgoing routes sent via FPVs. The contents of this table is a subset of the corresponding RibInTable of the neighboring router.
- *PeerData*: Contains information related to the peering link:
  - neighbor’s AS number and identification number (ID),
  - IP tuple of the connection,
  - neighbor’s peer type (customer, provider or peer),
  - metric, or cost of the link
  - various timeout values (hold, retry, keepalive)
- *HLPCore*: The HLPCore contains the Lib, DecisionTable, ExceptionTable, and the AS-Prefix Map. It maintains the periodic update timer<sup>3</sup>, and interfaces between the user and the system. The core also connects Peers with the Lib and DecisionTable, so that incoming routing information can be processed and then pushed out of the system if necessary.
- *Link-State Information Base (LIB)*: The Lib stores link-state information gathered from LSAs received. The network topology constructed is then used to determine the shortest path to each destination AS from every neighbor.
- *DecisionTable*: The DecisionTable chooses the winning route amongst the routes stored in the RibInTables for a particular destination prefix. The selection is based on the order of preferences given in Section 4.3. Winning routes are stored in a trie, after which they are pushed to the neighboring routers based on the route type as specified in Section 4.4. In general the DecisionTable deals with FPVs, whilst the Lib deals with LSAs.
- *ExceptionTable*: The ExceptionTable stores the exceptions raised locally, as well as those raised by other ASs within the same hierarchy (exceptions raised are not explicitly propagated to other hierarchies). Currently, only information with regards to exception 1 are disseminated via LSAs. The ExceptionTable is used when the DecisionTable is determining whether a particular winning route should be sent to a neighbor, and when the Lib is computing the shortest path to destination ASs taking into account exception 1s raised.
- *AS-Prefix Map*: This object stores the mapping of ASs to the corresponding advertised prefixes. Since the core of HLP manages routes at the prefix level<sup>4</sup>, and route changes are disseminated at the AS level, the AS-Prefix Map is required to translate between the two. Thus, for instance, incoming route changes (which will not include the prefixes involved, but just the AS path) will be processed in the HLPCore at the prefix level, and merged again just before the updates are pushed out.

## 7 Finite State Machine

The finite state machine for each peering connection is the same as that specified in [2].

<sup>3</sup>Updates can be either triggered or periodic. Periodic updates reduce the number of routing messages sent.

<sup>4</sup>Management of routes at the prefix level eases the migration of BGP software to HLP, since BGP currently operates at the prefix level. It is also flexible, allowing for traffic engineering at the prefix level, although this is currently not allowed

## 8 Protocol Format

The message header format as specified in [2] remains unchanged, as is the format of the Keepalive packet. In this section we describe the changes to the Open and Update (renamed Fragmented Path-Vector) packets, as well as introduce the LSA packet.

### 8.1 Open Packet

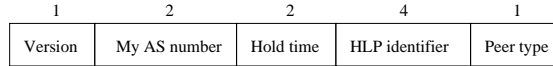


Figure 7: Format of Open packet, numbers denote lengths of corresponding fields in octets

Since a HLP network is dependent on the relationship between neighboring ASs, it is important that they agree on that. We thus include an additional field in the Open packet, the *peer type* field. The relationship type inserted in the field is with respect to the neighbor. For instance, if AS A is a customer of B, it will insert type corresponding to customer in the field of the Open packet it sends to B. Inconsistencies will result in the connection failing.

### 8.2 Fragmented Path Vector Packet

The format of the FPV, shown in Figure 8, is similar to BGP’s Update packet, but with an additional AS Down field. When an AS becomes unreachable for any reason, rather than withdrawing every route to that AS, we propagate just the AS number in this field.

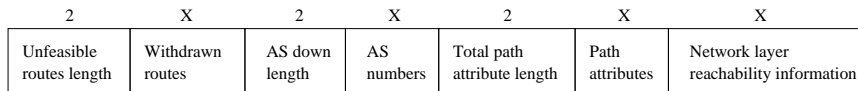


Figure 8: Format of FPV packet, the numbers denote lengths of corresponding fields in octets. An X means that the field length is variable.

The format in which a network prefix is represented is shown in Figure 9. This representation is used for the network layer reachability information (NLRI) and withdrawn routes in Figures 8 and 10.

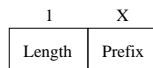


Figure 9: Representation of a prefix: the Length field uses 1 octet, and the size of the Prefix field is variable.

### 8.3 Link State Advertisement Packet

The LSA packet is a new packet type, and the fields are shown in Figure 10.

We describe the four main fields and their corresponding subfields as follows:

1. *link changes*: This major field contains information regarding links grouped together according to an endpoint AS’ number. For instance, all links stated in “link information (1)” have an end AS with number “AS (1) number”. Thus, multiple link changes can be aggregated and transmitted within the same packet. The format in which information for each link is transmitted is shown in Figure 11.
2. *unreachable ASs*: These fields provide the list of ASs (“Unreachable ASs”) that are declared unreachable from an AS (“AS unreachable from”). This field is used to disseminate an AS’ setting of exception 1.

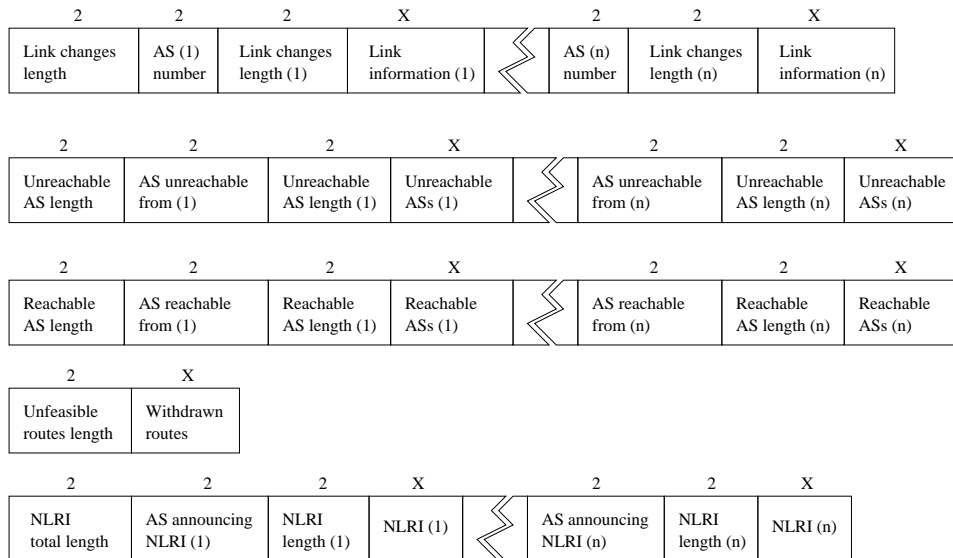


Figure 10: Format of LSA packet. The numbers represent field size in octets, X means the field size is variable.

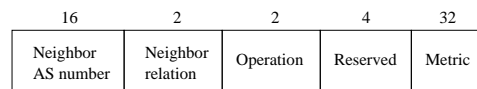


Figure 11: Link information representation format. The numbers represent field size **in bits**.

3. *reachable ASs*: These fields provide the list of ASs (“Reachable ASs”) that are declared reachable from an AS (“AS reachable from”). Note that the list of ASs must have previously been declared unreachable. This field is used when deleting exception 1.
4. *unfeasible routes*: Similar to Update packets, this field holds the routes that have been withdrawn by ASs in the same hierarchy.
5. *NLRI announcements*: The final field gives the prefixes that each AS is announcing.

## 9 Boot Up Procedure

The HLP protocol does not require knowledge of the tier level an AS is at. When connection to a new neighbor is established, the following steps are taken:

```

if neighbor is a peer
  send all routes tagged with PEER_FPV and CUSTOMER_LSA
if neighbor is a customer
  send all routes tagged with PEER_FPV and CUSTOMER_LSA
  send all link-state information
  send all exception information
if neighbor is a provider
  send all link-state information for customer ASs
  send all exception information

```

Additional routes that match exceptions, if any, are also sent.

## 10 Exception Setting and Removal

HLP allows dynamic setting and deletion of exceptions. Setting of exceptions should cause the network state to become the same as if the exceptions were present on bootup. Similarly, deletion of exceptions should cause the state to be the same as if the exceptions were never present. The following steps are taken when the corresponding exceptions are raised or removed:

- *Exception 1:* Upon setting of exception 1, the AS broadcasts an LSA packet in its hierarchy indicating the lack of a customer route to the destination AS. It removes route(s) to the destination AS (tagged with CUSTOMER\_LSA), and disseminates “AS down” to the relevant neighboring ASs via FPVs. When an exception 1 is removed, the AS broadcasts reachability of the destination AS via itself, recomputes the shortest paths’ costs, and disseminates the relevant FPVs.
- *Exception 2:* The setting of exception 2 is not disseminated explicitly in LSAs. The AS concerned iterates through the winning routes in the DecisionTable, selects the routes originating from the provider (both PROVIDER\_LSA and PROVIDER\_FPV) specified in the exception, then pushes the routes out the peer (also specified in the exception). When the exception is removed, the AS announces the withdrawal of routes to the peer (typically in the form of “AS downs”).
- *Exception 3:* Similar to exception 2, setting of exception 3 is not disseminated explicitly. Winning routes from the peer specified is pushed to the provider. Removal of the exception causes withdrawal of the same routes.

## 11 Conclusion and Acknowledgments

The implementation of HLP hard-codes typical route forwarding rules in BGP. By extending a working version of BGP, it is hoped that the time taken to implement the protocol is reduced.

We would like to thank Pavlin Radoslavov of the XORP group for his assistance in this project.

## References

- [1] XORP, the eXtensible Open Router Platform, <http://www.xorp.org>
- [2] Rekhter, Y., Li, T., “A Border Gateway Protocol 4 (BGP-4)”, *RFC 1771*, T.J. Watson Research Center IBM Corp., Cisco, March 1995