Interdomain Traffic Engineering in a Locator/Identifier Separation Context

Damien Saucez, Benoit Donnet, Luigi Iannone, Olivier Bonaventure
Université catholique de Louvain, Belgium

Abstract—The Routing Research Group (RRG) of the Internet Research Task Force (IRTF) is currently discussing several architectural solutions to build an interdomain routing architecture that scales better than the existing one. The solutions family currently being discussed concerns the addresses separation that scales better than the existing one. Such a separation provides opportunities in terms of traffic engineering. In this paper, we propose an open and flexible solution that allows an ISP using identifier/locator separation to engineer its interdomain traffic. Our solution relies on the utilization of a service that transparently ranks paths using cost functions. We implement a prototype server and demonstrate its benefits in a LISP testbed.

I. INTRODUCTION

During the last years, the Internet growth combined with factors including multihoming and interdomain traffic engineering has lead to a huge growth of the BGP routing tables ([1], [2]) and an increase of the BGP churn [3]. To cope with this problem, the Internet Architecture Board (IAB) rechartered the Routing Research Group (RRG) of the Internet Research Task Force (IRTF) to design a new interdomain architecture for the Internet. The architectural work within the RRG is progressing slowly, but several proposals have already been discussed ([4], [5], [6]).

Most of these proposals assume two different types of addresses: identifiers and locators. An identifier is used on an host to identify a connection endpoint while a locator refers to a node attachment point in the Internet topology. Note that, in today’s Internet, an host address is at the same time its identifier and its locator. The proposals are divided in two categories: those attaching locators directly to hosts (see HIP [7], SHIM6 [8], or ILNP [6]) and those attaching locators to routers (see LISP [4], or Six/One [5]). Finally, when identifiers are not routable, a mapping system allows to map an identifier onto a set of locators in order to reach this identifier ([9], [10]).

A key advantage of the addresses separation is to offer the possibility of associating several locators to a given identifier. This implies the availability of multiple paths between two identifiers and, as shown by several studies, those paths often offer different characteristics ([11], [12], [13], [14]). Consequently, the identifier/locator separation adds a new dimension to traffic engineering. Indeed, the separation makes possible to choose the best locator (and, thus, the best path) in addition to current traffic engineering techniques.

This work was partially supported by the European-funded 027609 AGAVE, 034819 OneLab and INFSO-ICT-216372 TRILOGY projects.

In this paper, we propose a service helping an ISP, using identifier/locator separation, to deploy a traffic engineering service controlling both incoming and outgoing packet flows. This service is contacted by clients (i.e., LISP routers in this paper) for ranking paths between locators of the source and the destination. The service performs the ranking using cost functions that return a value characterizing a path according to one or more metrics. One of the main advantages of the cost functions is that they can be combined in order to form a more complex function. Clients receive the list of ranked paths so that the first path in the list is the most preferable while the last one is the least desirable.

We implement our path selection mechanism in a tool named ISP-Driven Informed Path Selection (IDIPS) and apply it to a LISP case study. Our implementation is freely available.1

The remainder of this paper is organized as follows. Sec. II gives a brief overview of LISP, a solution for separating addresses; Sec. III details our path selection mechanism; Sec. IV shows how to construct a complex path selection algorithm based on elementary cost functions; Sec. V demonstrates the benefits of using our path selection mechanism within LISP; Sec. VI positions our work with the state of the art. Finally, Sec. VII concludes this paper by summarizing its main contributions and by discussing future research directions.

II. LISP

The Locator/Identifier Separation Protocol (LISP) is a router-based solution that does not require any end-host modification ([4], [15]). LISP has been mainly designed for stub ASes and we focus our discussion on such ASes.

In a LISP-enabled stub AS, only the border routers are upgraded to support LISP. The hosts send and receive IP packets using IP addresses. In the LISP terminology, these host addresses are called Endpoint Identifiers (EIDs). The EIDs are not advertised in the BGP routing system. The addresses of the LISP-enabled routers are called the Routing LOCators (RLOCs). These RLOCs are allocated by the providers to which each border router is attached. An EID can be associated to several RLOCs.

The basic idea of LISP is to tunnel packets from the RLOC associated to the source EID to the RLOC associated to the destination EID. To better understand the operation of LISP, let us consider the example shown in Fig. 1.

In this topology, host EIDx is reachable through two border routers. Hence it can be associated to two RLOCs: $\text{RLOC}_1^{\text{EID}_x}$ and $\text{RLOC}_2^{\text{EID}_x}$. 

1See http://inl.info.ucl.ac.be/idips.
Our assumption is that specialized boxes are installed in the network. These boxes are in charge of running a path selection algorithm reflecting the operator requirements in terms of traffic engineering. Every time an application or service needs to select one path among others or to rank a list of paths, it contacts the box that replies with ranked paths. In the IDIPS terminology, the specialized box is called a server and anything querying the server is called a client. It is worth to notice that more than one server can be deployed in the network and that clients do not have to deal with that (e.g., servers can be deployed in anycast).

Any request sent by a client contains the following information: a list of sources, a list of destinations and an optional performance criterion (e.g., route stability). The server processes the request and builds a list of all possible paths based on those two lists. This paths list is then ranked using information on the network state owned by the server such that the higher the rank the more promising the path.

Sources and destinations sent by the clients are typically IPv4 or IPv6 addresses. Instead of replying with complete addresses, the server can work with prefixes. In other words, if two paths with different ends have the same rank and if it is possible to aggregate the sources within a single prefix and to aggregate the destinations within a single prefix, the returned ranked list will only contain one path where the source is a prefix encompassing the two source addresses and the same for the destination. Aggregation offers several advantages. It allows to reduce the size of the replies (e.g., a single prefix can include several addresses indicated in the request) as well as the amount of potential paths to process. In addition, it avoids revealing topology details and network policies to clients. Nevertheless, a drawback in such an approach is the lost of precision (e.g., the reachability of a prefix is not the same as the reachability of an host).

In addition to a ranked prefixes pairs list, the server reply contains a time-to-live (TTL) information indicating how long the path ranking remains valid. This TTL is configured by the network operators and depends on the performance criterion provided by the client. When the TTL expires, it is up to the client to contact the path selection service to obtain a new path ranking.

Considering ranked prefixes pairs list allows reduces the risk of attacks or the disclosure of sensitive information to competitors, which is often a required by ISPs. Further, it allows the operators to modify the ranking algorithms according to their needs without involving clients. It thus separates the clients and operators while enabling cooperation.

### B. A Modular Architecture for IDIPS

Fig. 2 shows the high level design of IDIPS, our implementation of the path selection service described in Sec. III-A. It is based on three modules or engines that cooperate with each other when performing the path selection.

The first engine, named Path Information Collector (Pic) collects path information. Information are of two types: (i) administrative information (i.e., network policies and billing), but also routing information such as BGP or IGP, (ii) measurements information (i.e., active and passive measurements).
In addition to information collection, the Pic translates the different metrics into path attributes. Attributes are a generic representation of the metrics, independent of their nature. The simplest way to transform metrics into attributes is to convert them into integer values. This idea comes from the LocalPref attribute used by BGP where complex metrics are summarized as an integer. Attributes comparison relationship is transitive so that the comparison between different unrelated paths is made possible. For instance, if $A > B$ and $B > C$ then $A > C$ for a given attribute. It is up to the implementer to check that the transitivity property holds.

Once paths have been characterized, their attributes are stored in the Knowledge Base (KB). The Knowledge Base might be seen as a database gathering all attributes of various paths. The Knowledge Base must face two main challenges. First, it must be possible to retrieve any path attribute as quickly as possible. Second, given the potentially large number of paths and attributes in the Knowledge Base, the Knowledge Base must be as compact as possible.

Finally, the Decision Engine (DE) compares the paths in order to select the best one according to some criteria. To do so, the Decision Engine defines Cost Functions. A Cost Function returns the cost of a $(source, destination)$ pair (i.e., a path) for a given criterion. The cost is a numerical value characterizing a path according to one or more metrics. The cost must respect two constraints. First, the lower the cost, the better the path. Second, costs comparison relationship has to respect transitivity. As for attributes, transitivity is the key point of Cost Functions as it allows one to estimate the cost of any path independently and then order them afterwards. Transitivity allows caching and parallel computation of costs. Another important point of Cost Functions is enabling combinations to create more complicated Cost Functions.

To ranks paths, the Decision Engine calls the appropriate Cost Function for each possible path to rank. It then creates the ranked paths list such that the best paths are those with the lowest cost and the worst with the highest. Paths in the returned list are grouped by rank. The first group of paths in the list contains all the paths with the same lowest cost value. The second group contains those with the second lowest cost and so on. Remember that topology should not be revealed, thus ranking are not absolute but relative to other paths in the list. For instance, if paths $A, B, C$ and $D$ have a cost of $1, 4, 1$ and $7$ respectively, the ranking value should be $A : 1, B : 2, C : 1$ and $D : 3$ which does not reveal the cost of the paths.

### C. IDIPS and LISP Interactions

In the LISP context, for each RLOC mapped to an EID, the mapping system provides a priority and a weight [4]. When several RLOCs have the same priority, the LISP traffic is split among the different RLOCs in proportion to their weight. This makes possible to control the traffic that enters a site by tuning the RLOCs sent to different sources and also by changing their priorities and weights.

The priorities in LISP follow the same principle as the ranking in IDIPS. It is thus straightforward to use the IDIPS ranking as an input for the mapping priorities. When RLOCs priorities have to be determined, IDIPS is contacted. The ranking in the IDIPS responses is then converted into priorities.

Working with prefixes in IDIPS allows one to reduce the number of exchanged messages. If IDIPS replies with prefixes instead of addresses, it becomes possible to use a reply for different EIDs. For instance, if an EID has RLOCs within prefixes already returned by IDIPS, it is possible to reuse the ranking. This technique has two advantages. First, it reduces the traffic to and from IDIPS and, second, it reduces the time required to obtain the optimal RLOC priorities. A drawback is that a specific cache has to be implemented on the LISP router.

### IV. Cost Functions

In this section, we show how to construct a complex path selection algorithm based on elementary Cost Functions. We base our explanation on a situation in which an ISP has three customer families: (i) premium users always requiring the best available performances, (ii) standard users requiring a good performance/cost trade off, and (iii) light users always requiring the lowest cost. The traffic engineering changes between the night and the day for standard users: during the day, a lower cost is preferred while during the night, the performance is preferred. The monetary cost of a path depends on the 95th percentile load of the link used to reach the Internet.

In our example, we assume that the function `update_prefix(src, dst, a, v)` tags path from `src` to `dst` with value `v` for attribute `a`. In addition, function `path_attributes(src, dst)` returns all the attributes of the path from `src` to `dst`.

We first have to define if a destination is reachable or not from a given source address. A destination is considered unreachable if its DISABLE attribute is set to 1 in the Knowledge Base or if it is impossible to find a longest-match prefix with at least one attribute. The Cost Function `is_reachable_cf`, implemented in Algorithm 1, returns 1 if the path is valid, 2 otherwise, so that reachable paths are preferred.

In our example, we assume that the local ISP is charged on the 95th percentile link utilization by its upstream ISPs. In addition, we assume that the local ISP receives one RLOC per upstream ISP. To support 95th percentile in IDIPS, a
monitoring tool estimates the cost of using the links at time $t$. Periodically, this monitor contacts the PIC to update the Knowledge Base with a cost associated to each upstream ISP. If the RLOC provided by upstream ISP A is $192.0.2.1$ and the estimated cost is $1,500.00$, DIDPS is updated as follows: update_prefix($192.0.2.1/32$, $0.0.0.0/0$, ‘COST’, $2$). It adds the attribute COST with the value $2$ for any destination with a source address within the prefix provided by ISP A. Here, the value of the attribute COST represents the ceiling cost of using the ISP, in kilo dollars. This approximation permits to avoid oscillations and can be adapted to the needs of the DIDPS operator.

Algorithm 2 shows the minimize_cost_cf cost function that returns the cost of using a link such that the cost at the lowest price is preferred.

When considering bandwidth, the best paths are those having the highest available bandwidth. To support available bandwidth metric in DIDPS, we add the ABW attribute representing the rounded available bandwidth expressed in Kbps on the associated path. The declaration of the available bandwidth (let say $12.5$Kbps) from src to dst can be done as follows: update_prefix(src, dst,’ABW’, $12$).

The implementation of a cost function preferring paths with the highest bandwidth is not straightforward. Indeed, DIDPS, by definition, always prefers the lowest cost while in terms of bandwidth, the highest is the best. Thus, to prefer paths with the highest bandwidth, the value of the available bandwidth is subtracted to the highest theoretical available bandwidth for the operator (i.e., the capacity of the best link in the network). Algorithm 3 provides the implementation of such a Cost Function, $\text{MAX BW}$ being the highest theoretical available bandwidth for the operator.

To implement the path selection algorithm, we need to define the customer family (i.e., premium, standard, light). For the example, we assume that prefixes are grouped in families. We add the FAMILY attribute to the Knowledge Base. For a customer belonging to the premium family (assuming that the associated EIDs are $192.0.2.224/27$), the Knowledge Base can be updated as follows: update_prefix($192.0.2.224/27$, $0.0.0.0/0$, ‘FAMILY’, $1$). Standard family has value $10$ and light family $20$. This family representation offers high flexibility (e.g., the family can change from destination to destination).

Like for cost minimization, the customer family cost function only has to return the customer family for traffic from src to dst. The previous algorithms can be combined by the network operator to build more complex strategies. Algorithm 4 shows the implementation of this Cost Function.

The previous algorithms can be combined by the network operator to build more complex strategies. Algorithm 5 combines all the blocks in order to reflect the operator policies proposed earlier in this section. In particular, Algorithm 5 first checks whether a path between src and dst exists. If at least a path exists, then it applies the policies previously defined, based on the on the FAMILY attribute. For premium clients available bandwidth is always preferred. For standard clients the applied policy depends on the time period; the available bandwidth is used as cost function during the night, while cost minimization is preferred during the day.
In this section, we evaluate the benefits of the interaction between LISP and IDIPS. To do so, we build the testbed depicted in Fig. 3. The left hand network, labeled Content Producer, is a content producer and the right hand network is the consumer. Interdomain connectivity is ensured by LISP. For the test, we used the two types of customers light and premium and apply the customer_management.cf cost function presented in Sec. IV. As discussed in Sec. IV, the objective for light users is cost reduction. On the contrary, QoS has to be ensured for premium users. In the sake of clarity, in our experiments two clients with one flow per client are involved. The light client downloads a large file using FTP (TCP) while the premium client watches a video over UDP.

The video must have at least a 1.4Mbps bandwidth and the jitter must be limited. The two networks are connected with two links: L1 and L2. L1 represents a peering link and L2 a customer/provider link (from the producer point of view). L2 is protected by a 128Kbps backup link. Penalties are due when QoS is not ensured for premium users.

In the testbed, we use the recent LISP implementation named OPENLISP [18]. OPENLISP implements the LISP protocol in the FreeBSD kernel. A particularity of OPENLISP is the mapping socket that allows user space applications to interact with the EID-to-RLOC mappings maintained in the kernel.

An IDIPS server instance runs in each network. At that point, neither OPENLISP nor IDIPS are aware of each other. The link between the two paradigms is implemented by a wrapper running on each OPENLISP router. The wrapper monitors the mapping socket and the IDIPS control plane. When there is an event concerning an EID of the local OPENLISP’s map table, the wrapper retrieves all the RLOCs for that EID and asks the IDIPS server to rank them. The resulted ranks are translated into priorities and the EID’s mapping is updated according to the information given by IDIPS. Our implementation does not yet take into account the LISP weight defined in LISP.

The experiment is divided in four periods (P1 to P4). The RLOCs used for each period depends on the IDIPS rankings. During P1, both L1 and L2 are working properly and IDIPS optimizes the performance for premium traffic and minimizes the cost for the light traffic. The beginning of P2 corresponds to the L2 link failure: L2 traffic is diverted to the backup link. During L2, IDIPS is not involved and the RLOCs are not modified, premium traffic is degraded. In P3, IDIPS is informed of the failure and modifies the mapping to minimize the cost and avoid backup links. It is worth to notice that the gap between P1 and P3 is for illustration only, in practice, IDIPS can be informed of the topology change at the same time as the backup link activation (e.g., via SNMP). During P3, the backup link is not used anymore. Finally, in P4 IDIPS decides to divert the light traffic (i.e., FTP) to the backup link and keep premium traffic on L1 to ensure its QoS requirements while minimizing costs.

Fig. 4 shows the flows’ dynamic during the different periods. The horizontal axis is the normalized time and the vertical axis the bandwidth (in Kbps). The best effort traffic consist of a big file transfer using FTP (TCP). The video is simulated with Iperf. Iperf continuously sends 700 bytes long UDP segments with a constant rate of 1.7 Mbps. During P1, both flows are working as expected: the video (premium customer) encounters a limited jitter and has enough bandwidth (1.7Mbps) and the cost for FTP (light customer) is minimized. After the failure, during P2, the video stream is redirected to the backup link. The video flow bandwidth falls down to around 100Kbps, which is not sufficient to ensure QoS (1.4 Mbps is required to ensure QoS requirements). FTP traffic is not affected by the failure as it is carried by L1. P3 presents the flow bandwidth when all the traffic is diverted on L1. For that period, the policies in IDIPS are to avoid backup links. However, this choice does not ensure QoS for the video as the jitter is important and video bandwidth falls to 1.3Mbps. With this configuration, video traffic is influenced by the TCP behavior of the FTP flow. Period P4 shows what happens if IDIPS is configured to ensure QoS and minimize costs, thus video is diverted on link L1 as this is the only one allowing QoS for video. The best effort flow is diverted to L2 backup link because the costs of using it is lower than the cost of losing QoS for video.

This test shows that IDIPS path selection algorithm can take administrative and technical question into account (e.g., minimize costs but maximize bandwidth). Furthermore, it also
shows that the simplicity of IDIPS allows to use it in situations where several paths are possible.

VI. RELATED WORK

A proposal that shares objectives similar to IDIPS is Morpheus [19], which determines the best path to use according to the operator policies and, then, sends BGP updates to its BGP router target (via multihop eBGP). Like IDIPS, Morpheus is very modular but is restricted to BGP as the signaling is performed using BGP messages while IDIPS has its own messaging format, allowing a finer-grained interaction between the client and the path selection service. Other IDIPS-like solutions have already been proposed, specially for peer-to-peer (P2P) applications. For instance, Xie et al. propose P4P, a solution to give topology hints to P2P clients [20]. In P4P, ISPs, or third parties, maintain trackers that are contacted by P2P clients to retrieve the best swarm peers according to some topology information. Aggarwal et al. introduce an oracle service very similar to IDIPS that would be configured by the network operator and queried by P2P applications [21]. While the oracle limits the ranking to destination addresses, IDIPS proposes to rank paths or even cluster of paths, using prefixes. Another difference between the oracle and IDIPS is the ranking scope: the oracle ranking is limited to local or peering domains while such a limitation does not exists in IDIPS. Finally, a number of vendors have proposed proprietary path selection solution ([22], [23], [24], [25]). Most of the proprietary solutions are based on BGP or NAT or rewriting.

VII. CONCLUSION

Identifier/locator separation proposals, such as LISP, are currently being discussed as a possible solution to better scale the Internet architecture. The idea is to assume two different types of addresses: identifier (i.e., a connection endpoint) and locator (i.e., a node attachment point in the Internet). As several locators can be attached to a given identifier, it leads to the increase of the number of available paths between two identifiers. It has been demonstrated that those paths often offer very different performance characteristics.

This paper proposed what is, to the best of our knowledge, the first attempt for performing traffic engineering in an identifier/locator separation context. Our solution, named IDIPS, allows an ISP to control both incoming and outgoing traffic without the drawbacks of traditional traffic engineering schemes.

IDIPS is a service deployed in a stub AS network using LISP. It is contacted by border routers performing the identifier/locator mapping to rank paths. These rankings are performed using cost functions that return a value characterizing a path according to one or more metrics. One of the main advantages of the cost functions is that they can be combined in order to form a more complex function. Clients receive the list of ranked paths so that the first path in the list is the most preferable while the last one is the least desirable. We demonstrated the advantages of coupling LISP and IDIPS through a case study.

The path ranking service provided by IDIPS is applicable to the other environments where a host or a set of hosts are reachable via multiple paths such as IPv6 host based multihoming [8], IPv4/IPv6 dual-stack hosts or peer-to-peer applications. In the near future, we plan to evaluate the effects of IDIPS on real traffic.

REFERENCES