FIB Aplasia through Probabilistic Routing and Autoforwarding

Giuseppe Rossini[a,1], Dario Rossi[a], Christophe Betoule[b], Remi Clavier[b], Gilles Thouenon[b]

[a] *Telecom ParisTech, Paris, France*
[b] *Orange Labs, Lanion, France*

Abstract

In this work, we propose APLASIA, an holistic architecture with a radical design. Aiming at simplifying the inner network devices (and so their cost), we tradeoff node architecture- and algorithmic-complexity for an increased (but tunable) communication cost. The main ingredients of our recipe are (i) the use of complete paths directly in the frames header, that allows core devices to perform data-plane switching functions without lookup and (ii) the use of a greedy probabilistic routing algorithm to quickly discover multiple, near optimal, paths in the control plane. We extensively simulate, analyze and implement our proposal to testify its soundness.

1. Introduction

In the last few years, several research proposals have addressed the issue of compact routing at the Intranet [1, 2, 3, 4] or, more recently, at the Internet[5, 6, 7] level. Often, these architectures employ traditional routing protocols for their inner-working: as link-state algorithms, such as OSPF or IS-IS, are about 20 years old, they reveal their limits when used in flat environments.

In this work we present a new holistic proposal for the underlying multi-path routing of the above architectures, especially targeting *intra-domain* routing. Our proposal sits at a radical point, unexplored so far, in the network design space, where we tradeoff hardware and algorithmic simplicity for a slightly increased (but tunable) communication cost.

We name our proposal *Adaptive Probabilistic Link-state Architecture Switching in Autoforwarding* (APLASIA). In medical terms, Aplasia refers to a congenital absence of an organ or tissue: in our architecture, this condition refers to the absence of a Forwarding Information Base (FIB) in the switching fabric of core network nodes. We tradeoff the absence of FIB information with a minimal increase of frame header size: in an APLASIA domain, data frames carry a fully specified source-routed path, so that the next hop interface is always directly available in the frame header (i.e., data frames are *autoforwarding*) and does not require any lookup.

We believe Aplasia to be an advantageous condition. Our reasoning is that data-plane autoforwarding simplifies the nodes hardware architecture (and their cost), as it eliminates the need to implement FIB data structures able to cope with link speeds and thus requiring fast and expensive memories (e.g., TCAM for longest-prefix match lookup in the IP world, or CAM for Ethernet switching).

Additionally, to further simplify the nodes inner architecture, we reduce the computational complexity of the algorithm run by control plane software (further limiting the hardware requirements in turn), that we again tradeoff for an increased (but tunable) communication cost. Under link-state algorithms such as OSPF and IS-IS, topology discovery is performed by running the Dijkstra algorithm after each node has received all link-state advertisements. In APLASIA, as the control-messages accumulate the traveled path directly on the header, there is however no need to run additional algorithms for path discovery, that can be instead be learnt and updated on each new control message reception.

Our proposal simply reinforces link-state advertisement with a greedy *adaptive probabilistic component*, that builds up on our previous work[8], and generates an additional bounded number of messages, controlled through a single parameter that tunes the tradeoff between path quality and extra messages cost. The algorithm we propose is not only simple and greedy, but is also able to find multiple paths that are (i) loop-free and (ii) near-optimal, where the optimality criterion is the

[1] Corresponding author giuseppe.rossini@enst.fr

disjointness of the paths. Furthermore, our results show path quality to degrade gracefully in case of parameter misguidance.

This work extends [8] in several directions: from the algorithmic point of view, we make an analytical cross-comparison with other routing architectures, and simulate the algorithm on bigger realistic networks. Then, we precisely model the duration of a single advertisement round showing as the model fits well simulation results. Finally, some practical considerations are made in order to improve the quality of the paths discovered (e.g., in terms of path disjointness).

From the architecture point of view (never discussed in [8]) we accurately design and prototype the forwarding plane of APLASIA. Such design is carried by the practical observation that the original algorithmic solution is well suitable for a source routed forwarding.

The remainder of this paper is organized as follows. In Sec. 2 we place our work in the context of related effort. Then, Sec. 3 details APLASIA's two main components, namely (i) switching in autoforwarding and (ii) the multi-path adaptive probabilistic link-state routing algorithm. We investigate APLASIA performance with simulative and analytical techniques: we focus on properties such as the path-quality vs message-cost tradeoff (Sec. 5) and timeliness (Sec. 6). We then report on our Click implementation in Sec. 7, and conclude the paper in Sec. 8

2. Background

The tradeoff between the amount of state needed for routing and forwarding functions, and the quality of the gathered communication path has been explored at all layers of the protocol stack and contexts – from application-layer peer-to-peer overlays, to Internet intra- and inter-domain routing and to lower-layer access architectures at the network edge.

In the current state of affairs, for instance, IP attempts at solving scalability via hierarchical addressing and aggregation. However, hierarchy requires location-dependent names that complicate network management (e.g., mobility) already at the IP level. Location-dependent names can provide enough benefits to become appealing only for specialized environments (e.g., as in PortLand[9], that leverages positional addressing to optimize switching on data-centers arranged as a fat-tree topology). For more general purposes, location-independent (or *flat*) names have received a growing interest lately (see [7] for a thorough overview).

While in principle such flat identifiers can be arbitrary bit-strings, Ethernet MAC addresses represent perhaps the best example. Indeed, as Ethernet is the most widely deployed fixed access technology as of today, much research effort focused on making Ethernet scalable, with relevant research proposals (such as SmartBridges [1], Rbridges [2], Viking [3], SEATTLE [4], SPAIN [11]) and normalization effort grouped under the "carrier grade" Ethernet umbrella [12] (of which examples are IEEE 802.1ah PBB[13] and its traffic engineering extension IEEE 802.1Qay PBB-TE[14]).

We summarize and compare relevant related effort in the above areas in Tab. 1, where we list some among the proposal for scalable routing on flat identifiers from an Internet[7, 6, 5] or Ethernet perspective [1, 2, 3, 4, 11] (the latter being closer to our work given our focus on intra-domain routing). We point out that while each of these two domains has its own specificity, architectural solutions can be shared to some extent. Let us focus on *forwarding state scalability* first. For instance, in the Ethernet context, one of the issues concerns the possibly very large number of hosts – which affects the amount of state that switches have to keep on the one hand, and the procedure used for the host resolution on the other hand. In Ethernet, the space of existing solutions ranges from MAC address encapsulation (as in IEEE 802.1ah PPB, so that only the outer MAC addresses, but not the inner host MAC, are exposed within the domain), to the use of distributed hash tables (as in SEATTLE [4] using a One-hop DHT to perform scalable host resolution). Similarly, ROFL [6] applies techniques from DHTs, though it targets Internet intra- and inter-domain routing.

Let us now consider the issue of *routing and path quality*. In the case of Ethernet, another long studied issue concerns the limits of the Spanning Tree Protocol (STP), whose main purpose is to provide loop free paths between any two nodes in the LAN: here, research has focused on a more efficient use of LAN resources (e.g., by the use of multiple spanning trees) and a faster convergence in case of topology changes or failures. To get around these limits, again the explored design space is rather wide, with solution ranging from centralized approaches [3], to the use of IETF GMPLS[15] control plane protocol suite to configure PBB-TE devices, or the use [4] of classical link-state routing protocols such as OSPF. Yet, even in this case some commonalities can be identified across domains of application, as at the Internet level ROFL [6] still assumes an underlying OSPF-like protocol to detect link and node failures, while Disco [7] resorts to a Distance Vector (DV) algorithm to propagate addresses.

APLASIA operates at an unexplored point in the routing-state vs path-quality tradeoff, as it shifts the

Table 1: Comparison of related effort

Architecture	**Host resolution**	**Routing**	**Algorithm complexity**	**Communication complexity**	**Multipath**
Rbridges[2] SEATTLE [4] ROFL[6]	Centralized One-hop DHT Chord DHT	LS (OSPF)	$O(NlogN + N\delta)$	$O(N\delta)$	No No No
SmartBridge[1]	Centralized	Diffusing computation + BFS	$O(N\delta)$	$O(N\delta)$	No
BANANAS [5]	n.a.	LS + k-shortest path[10]	$O(N\delta+NlogN+kN)$	$O(N\delta)$	Yes (k)
Viking [3]	n.a.	Centralized + multiple runs of [10]	$O(N^3logN + N^3\delta)$	n.a.	Yes (2 out of k)
APLASIA	n.a.	APL	$O(N\delta)$	$O(N\delta)$	Yes (2)

routing state from within the core network devices to inside the header of messages traveling in the network. In this work, we do not focus on the name resolution issue that can be handled, e.g., as in [4]. Rather, we focus on two system aspects, namely forwarding and distributed routing protocol, that are necessary to enable the state shift.

2.1. Forwarding and framing

APLASIA employs source routed paths, with a peculiar path encoding that (i) eliminates the need for forwarding tables altogether, by fully specifying the sequence of output ports directly in the frame header. Another nice property of APLASIA framing is the ability to provide (ii) loop-free paths by design. This is as opposite to what generally happens in distributed algorithms, where loop-free properties have to be enforced by the protocol, and are handled e.g., as in DSR[16] by a TTL field (whose setting is however topology-dependent and may thus compromise the quality of the resulting paths) or as in DUAL[17] via diffusing computation techniques.

Framing constitutes an architectural aspect that recent Ethernet evolutions (such as Q-in-Q, MAC-in-MAC, PBB-TE, etc. [12]) have dealt with. However, these approaches merely define new framing capability to enhance Ethernet scalability, but do not otherwise impact the forwarding function. Moreover, such architectures loose the holistic view of the older technology, and are no longer plug-and-play, requiring configuration via a management or control plane, so that we consider a more detailed comparison out of scope.

It could be argued that the idea of stacking multiple labels has already been used in other contexts, as for instance in MPLS. We point out that the label purpose and semantic drift however significantly from ours. Label stacking in MPLS is indeed generally used for flexibility and to overcome label space scarcity [18]. Hence, the depth of the label stack is generally limited, and a lookup table is still needed to locally translate the label into the corresponding outgoing port.

Closer work under this angle is represented by Pathlet routing[19], Disco[7] and BANANAS[5] in the wired domain, and by Dynamic Source Routing (DSR) [16] in the wireless one. All these proposals go in the direction of specifying the full (or portions of the) path in the packet header, which as stated in [7, 19] introduces low overhead compared to IPv6 headers (we will come back to the framing overhead in Sec. 3.2).

For instance [19] proposes to specify *portions* of edge-to-edge paths (but not full paths) in the header, to extend the routing policy expressiveness in the inter-domain context. BANANAS[5] instead routes along fully specified path sequences (composed by globally known nodes and interfaces identifiers) that are however compactly represented by fixed size *hashes*. During forwarding operation, a local table lookup is still needed to map local link indices to global link ID. However, the size of the table at each router is limited to $log_2 d$ as it depends on the local node degree d, but not on the network size N – which in medical terms already constitutes a beneficial atrophy of the FIB, but not a full aplasia.

Finally, in the wireless context, DSR [16] proposes to fully specify paths directly in the packet header. Differences from APLASIA lay here in the fact that, as the wireless medium is generally broadcast, the path can simply be identified as addresses instead of ports. Moreover, to avoid as possible to occupy the wireless medium with route discovery, DSR makes extensive use of route caching at intermediate nodes (and other techniques to automatically shorten the path if a node further away in

the unexpended portion of the path overhears a message reaching a preceding relay). Hence, DSR is engineered by considering the wireless medium a scarce resource –antipodean with respect to the abundance of capacity in APLASIA environments– leading to a completely different architecture design.

2.2. Routing

Broadly speaking, routing algorithms are classified as Distance Vector (DV) or Link State (LS) approaches. In traditional LS algorithms such as OSPF and IS-IS, a full knowledge of the network topology is however needed, which is gained by broadcasting local information among neighbors, after which well-know algorithms such as Dijkstra can be run to compute the shortest path to any given destination (or more complex algorithms to gather multiple paths). In DV algorithms, routing tables are updated incrementally at the reception of each message carrying global reachability information, while loops are resolved by diffusing computation[17] and multiple paths are also possibly supported[20]

Routing in APLASIA follows an Adaptive probabilistic link-state (APL) algorithm, where nodes propagate, as in LS, local connectivity information. At the same time, APL performs incremental distributed multi-path computation, and thus inherits some of the desirable DV properties. In APL each message accumulates the complete traveled path from the advertiser, so that paths can be updated at the reception of any new control message (as in DV) with a greedy algorithm. Moreover, APL supports the computation of multiple paths, via simple inspection of APL frame header and comparison operations (unlike in LS), requiring only a minimal amount of state.

While it is possible to limit the number of messages exchanged by APL to match LS message complexity (that already allows to learn multiple paths), however APL benefits of some additional exchanges (to ameliorate the quality of the additional paths). These extra messages are (probabilistically) sent in such a way that the APL procedure is not only auto-terminating (as in DV) and rapidly converging (faster than LS, see Sec. 6), but the number of messages sent over the whole network remains of the same order of magnitude (with a fixed, tunable, multiplicative overhead with respect to LS, see Sec. 5).

Probabilistic decisions are also used in [21], that employs distributed Ant Colony Optimization (ACO) algorithms, based on swarm intelligence. A set of ants is spread through the network in order to discover disjoint multiple paths, and the pheromone left by the ants is employed in order to probabilistically avoid already crossed paths. However, beside the increased algorithm complexity, we point out that the amount of (pheromone) state kept at each node scales as $O(N^2)$, whereas in $O(N)$ APLASIA[2].

Tab. 1 reports the computational and communication complexity of some relevant related work. Given a graph $G = (V, E)$, let us denote with $N = |V|$ the number of nodes in the graph and by $\delta = |E|/|V|$ the average degree. For comparison purposes, let us consider a classical LS approach, as this is used in close work[4]: on any topology change, the number of messages sent over the whole network is $O(N\delta)$, while the computation complexity of Dijkstra is $O(NlogN + N\delta)$. SmartBridges[1] achieve lower computational complexity but is still limited to a single (shortest) path forwarding. BANANAS [5] supports instead multiple paths: more precisely, after topological information is disseminated with an LS algorithm, nodes in BANANAS run a k-shortest path algorithm[10], that has a well known computational complexity equal to $O(NlogN + N\delta + kN)$ for finding k shortest paths to each of the N destinations.

Unfortunately, as paths found by [10] are however *not* guaranteed to be disjoint, this may not be sufficient neither for load balancing, nor for resilience. A solution to this problem comes from Viking [3], that however adopts a centralized incremental approach. The central node has first to (i) run a k-shortest path algorithm N times (one per each node), gathering k paths for all $(N-1)$ destinations, with computational complexity $O\left(N(NlogN + N\delta + kN)\right)$; then (ii) for each $N(N-1)$ source-destination pairs[3], it then needs to run a k-shortest path algorithm on a modified graph[4], each of which has a cost $O(NlogN + N\delta + k)$. Overall, the computational complexity for the central node amounts to (i)+(ii)$\approx O(N^3logN+N^3\delta)$ to compute a set of candidate paths (out of which only two are then selected). As reported in [3], this possibly leads to large execution times, on the order of tens to hundreds of seconds, on a

[2]Note that our currently ongoing work suggest that this could be bound to $O(1)$ under certain assumptions that we briefly discuss on the conclusion section.

[3]Notice that Viking consider that only an subset of nodes $N_S < N$ can be part of a source/destination pair, whereas the remaining $N - N_S$ nodes only perform switching functions and thus should not be accounted for in the algorithm complexity. While this also the likely application scenario for APLASIA, we however prefer to give computational complexity bounds for the general case.

[4]In the modified graph, rather than removing the links, the cost along the k shortest paths is increased by the original graph diameter: in this way, overlaps are tolerated only when strictly necessary, i.e., when the path would otherwise be disconnected.

8x8 grid even when only 4 out of 64 possible destinations are considered. Also SPAIN [11] points out that it is "computationally unfeasible to find the best path set of size k". Yet, the solution proposed in SPAIN is again centralized, though greedy and thus less computationally intensive w.r.t [3].

APLASIA aims at finding multiple, as disjoint as possible, paths with a greedy, simple and distributed approach. In this paper, we therefore compare the *quality* of the path found by APL against the centralized optimum computed as in Viking. However, as Viking follows a centralized approach, it does not make sense to directly compare computational and message complexity – rather, we take a classic LS approach as a term of comparison. As shown in Sec. 5, the communication complexity of APL remains $O(N\delta)$. It follows that, in case only a *primary* and a *secondary* paths are maintained, only $k = 2$ comparison operations are performed for each control plane message, so that the computational complexity per node remains $O(N\delta)$ as well.

Finally, we point out that in all LS-based approaches[5, 3, 4] a major drawback is that path computation occurs *after* dissemination of the topological information. Topological dissemination constitutes a first source of delay in the path construction process. Yet, gathering multiple paths can further slow down the process due to the growing computational complexity, which is especially true in case more sophisticated path properties are required (e.g., path disjointness as in [3] with respect to simpler k shortest paths in [5]). The fact that APLASIA allows to gather paths *during* topological dissemination constitutes a first advantage, the lack of a costly FIB update process[22] a second speedup, and the simplicity of the algorithm a last one.

3. Architecture overview

APLASIA offers a simple, multi-path, flexible, loop-free, fast, efficient, tunable and plug-and-play connectivity layer. To reach these goals, we take an holistic approach and jointly design new *forwarding* and *routing* mechanisms.

To simplify the forwarding in an APLASIA domain, *edge-to-edge paths are completely specified in the header* of each frame. Hence, forwarding decisions do not require a table lookup, as the next hop interface is carried in the header.

APLASIA supports *multiple paths* toward the same destination, that can be used in a flexible way on the data plane (i.e., for backup or load balancing). While the underlying APLASIA algorithm does not limit the number of paths[5] a priori, in this work we limitedly consider two paths for the sake of simplicity, and as ultimately selected in Viking [3].

Furthermore, our *adaptive probabilistic link-state* algorithm yields to a path creation process that is loop-free (by design), fast (quickly providing the primary and secondary paths), efficient (as it very often finds optimal paths) and tunable (in terms of the number of control messages overhead).

Finally, APLASIA operations rely mainly on a single parameter, to whose settings we offer guidance through analytical models, and that simulation results show to be non-critical in case of misconfiguration. Hence, APLASIA can be safely shipped with a default configuration, offering plug-and-play operations as it requires no configuration effort.

3.1. Node architecture

APLASIA nodes are addressed by flat identifiers that are univocally associated with them as in [6, 7, 4], and can be chosen irrespectively of nodes topological position.

Whenever a node intends to send data to another node, it needs to assembly a frame, specifying the complete APLASIA edge-to-edge path in the frame header (more details on framing in Sec. 3.2). The frame is then handed over to the data-plane for forwarding. As a complete path sequence is specified in the header, the frame is simply pushed to the output interface to which the Current Hop (CH) frame field points to ($CH = 0$ at frame creation, and is incremented by 1 at each hop). Moreover, all nodes along the path perform the same forwarding operation, so that no switching table lookup is performed in the data plane.

Roughly speaking, with reference to Fig. 1, we can identify two kind of nodes in the network, namely *edge* and *core* nodes: the former can be traffic sources, whereas the latter only performs switching functionalities. APLASIA follows the principle of pushing complexity to the edge of the network, so that core nodes need to keep only a (i) minimal amount of state (APL counters) to run the adaptive probabilistic link-state algorithm. Additionally, edge nodes store a (ii) cache of source routed paths (e.g., stored as an hash-table or as multiple-disjoint network slices as in [23]). It is worth pointing out that APL counters are used in the control plane only during the exchange of routing messages and that, similarly, the source-routing path cache is used by

[5] In practice, both load balancing and resilience [23] may benefit from the use of multiple $k > 2$ paths.

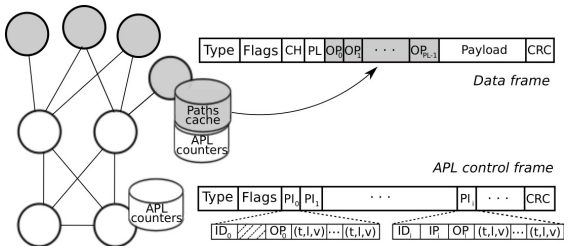

Figure 1: Synopsis of APLASIA control-plane routing state and sketch of data and control frame headers.

edge nodes only at frame generation time. Hence, these structures are accessed at a much slower rate with respect to the data plane forwarding operation deep in the core of the network, where a higher level of aggregation translates into higher speed.

3.2. Autoforwarding frames

While providing full details of APLASIA framing specification and encoding is out of the scope of this paper, we describe the main fields with the help of figure Fig. 1, sketching *data* and *control* frames.

Aside from usual fields such as frame type, node ID, flags, QoS indications and checksum, data frames carry an output path $\{OP_i\}_i$, that represents the set of output ports to follow edge-to-edge in the APLASIA domain, along with a CH pointer to the next output port and a path length field PL. By default, each OP_i consumes 8 bits in the header, which is *optimized* for nodes having at most 256-ports[6].

This choice has a number of advantages, the first of which is to simplify the rest of the architecture by stateless forwarding. This simplicity comes at the price of a (possibly) slightly increased overhead in the frame header, that grows proportionally to the path length. To make a rough comparison, consider for instance the amount of header space devoted to addresses in architectures such as IEEE 802.1ah PBB[13] or paths in BANANAS[5]: in the former, MAC-in-MAC frames carry four 48-bits MAC addresses for a total of 192 bits, while the latter considers 128 bits as "a reasonable bit budget" for encoding the path in the frame header. Sticking for the sake of simplicity to 8-bits individual OP_i, these bit budgets translate to 16-24 hops-long paths. Notice however that while in [13, 5] the 128- or 192-bits overhead is a fixed one, in APLASIA the overhead is instead *variable*, as it depends on the length[7] of the data-plane frames path.

To sustain the routing operations in APLASIA, the control plane makes use of APL frames. Instead of merely carrying information about individual links (as in classical link-state algorithms), APL frames accumulate information about *the whole traveled path*. While algorithms based on diffusing computation [20] already propose to complement link-state exchanges with information about the path (e.g., the second-to-last hop), avoiding loops within the network still remains a fairly complex task. In contrast, APLASIA pushes this trend even further: as paths are fully specified, it becomes *trivial to avoid loops* during path computation (Sec. 3.3).

With respect to data messages, control messages carry additional path information (PI). However, as the volume of control messages is low with respect to data exchanges, the overall control plane overhead is expected to be limited. In more detail, PI_i contains, besides output path OP_i, the corresponding node identifier ID_i (useful for *loop detection*) and input port IP_i (useful for *path inference*). The PI sequence grows at each hop during the path computation process. Additionally, optional information about the estimated path quality can be conveyed in the header, in the form of type-length-value (t, l, v) couplets in PI_i, to assist traffic engineering operations (out of the scope of this paper).

As control message carries path information, any edge node, handling or overhearing it, can infer topological information in a completely passive way: more precisely, any node receiving a control message that has already traveled i hops, can in principle learn paths to any of the previous $i - 1$ nodes up to the origin, that can (at node will) populate the source routing cache. As different messages possibly carry different information, multiple paths toward the same destination are actually found. Also, while in principle several criteria are possible for primary and secondary selection, for the remainder of this work the primary path is expected to be the shortest path (in terms of hops count). As secondary path, we instead retain the shortest path most disjoint from the primary: as this choice reduces the share of faith between these paths, it improves network resilience against both failures and traffic surges.

[6] APLASIA poses no limit to the number of ports per device. An header flag turns on *variable-size* output ports identifiers, whose size depends on a prefix-based notation, inspired by the old classful IP addressing: i.e., 8 bits for identifiers starting with `0` (allowing to address $2^7 = 128$ ports), 16 bits for `10` (allowing to address 2^{14} ports, and used only when the port identifier exceeds 2^7) and so on.

[7] Hence, in practice the overall overhead can be even *lower* with respect to [13, 5], since the path length for some data frames will likely be shorter than 16-24 hops.

3.3. Adaptive probabilistic link-state routing

Prior to describe the APL algorithm in detail, let us describe its main high-level idea. Upon reception of an advertisement message, nodes take flooding decisions independently for each of their output links, accumulating the previously discussed path information PI in frames header. The main idea is that nodes need to flood a received message *at least once*, so that shortest paths are discovered. But nodes actually need to flood the message *multiple times*, in order to ameliorate the quality of the paths found beyond the shortest one. The original idea in APLASIA is to cut exponentially fast the number of flooded messages, by retransmitting the message on each output interface (except the one from which the message has been received) with probability

$$P = \beta^{C_i} \tag{1}$$

where β is a fixed backoff parameter and C_i is a counter, locally stored by each node, of the number of times the node has already received an advertisement originated by i. After having flooded the message on its output links, C_i is incremented by one unit, decreasing the odds that the advertisement message of node i will be flooded at the next reception.

First, notice that in case we set $\beta = 0$ (and consider $0^0 = 1$), the number of messages sent by APL matches that of an LS routing algorithm. Hence, APL only sends a higher number of messages with respect to classic LS algorithm when $\beta > 0$. At the same time, the use of an exponential backoff drastically limits the number of transmitted messages (analytical bound in Sec. 5), making thus the approach scalable at the price of a minimal amount of state that needs to be maintained by all APLASIA nodes – namely, a set of APL counters associated to node identifiers (i, C_i). For simplicity, this structure can be thought as an hash of N byte-size[8] counters, with N the number of network nodes[9].

In more details, a source node s running the APL algorithm initiates the advertisement process by flooding an advertisement packet m to all its neighbors. The flooded message contains a list of node identifiers ID, initially set to ID[0]=s by the source, to which each node appends its own identifier. Upon reception of an advertisement frame m, in case the receiver j detects a loop (finding its identifier within the ID list, lines 4-5), it discards the message and aborts the flooding procedure.

If the node is a core device performing simple switching functionalities, then in virtue of the autoforwarding it does not need to learn source-routed paths (i.e., it can skip lines 6-15). Otherwise, if the node is equipped with a cache to store source-routed paths (i.e., is an edge node) it analyzes, and possibly stores, the newly learnt path $\mathcal{L}_{jd}$. Notice that, from a single message, a node may learn multiple paths, i.e., the backward paths to all intermediate nodes $m.ID[i]$ until the advertiser $s = m.ID[0]$. While this may be beneficial to speed up the learning process of newly joining nodes, it translates into a higher computational complexity whenever done systematically at each APL message reception (by a factor proportional to the path length, that can be upper bounded by the graph diameter). Hence, this feature is turned off by requiring $i = 0$ at line 6, but could be

APLS Algorithm

```
1: while { node j receives message m on interface rx } do
2:   ℓ ← length(m.ID)
3:   for all { i ∈ [0, ℓ] } do
4:     if {m.ID[i] = j} then
5:       exit // Break loop and abort flooding
6:     else if { node has source-routing path cache
       ∧(i = 0) } then
7:       d ← m.ID[i] // Destination
8:       L_jd ← (rx, m.IP[ℓ], ..., m.IP[i])
9:       if { ∄P_jd ∨ |L_jd| < |P_jd|} then
10:        P_jd ← L_jd // Update primary path
11:      end if
12:      if {∄S_jd ∨ (|P_jd ∩ L_jd| < |P_jd ∩ S_jd|) ∨
         (|P_jd ∩ L_jd| = |P_jd ∩ S_jd| ∧ |L_jd| <
         |S_jd|)} then
13:        S_jd ← L_jd // Update secondary path
14:      end if
15:    end if
16:  end for
17:  m.ID[ℓ + 1] ← j
18:  m.IP[ℓ + 1] ← rx
19:  s ← m.ID[0] // ID of the advertisement source
20:  for all {next ∈ interfaces(j)\rx } do
21:    m.OP[ℓ + 1] ← next
22:    send m to next w.p. β^{C_s}
       // Adaptive probabilistic transmission
23:  end for
24:  C_s++ // Update counter associated with node s
25: end while
```

[8] As (1) decreases exponentially fast, a wider counter is not necessary: consider that even for a backoff as high as $\beta = 0.95$, it is unlikely that control messages will be forwarded more times than a byte-size counter can handle (β^{255} is on the order of 10^{-6})

[9] At the same time this is unnecessary, since due to the autoterminating property of the APL procedure, the counter set could be implemented as a simpler FIFO structure with a small amount $M << N$ of elements (not discussed further for lack of space)

Network	Segment	N	δ	σ_δ	$\Delta[ms]$	D
Qwest	Core	33	5.0	3.1	5.9	5
DTelekom	Core	68	10.4	13.3	17.2	3
Level3	Core	46	11.7	10.1	8.9	4
Sprint	Core	315	6.2	6.9	3.2	13
Geant	Aggr	22	3.4	1.4	2.6	4
Tiger2	Metro	22	3.6	0.6	0.1	5
Random	-	$10^{[2,5]}$	4	≈ 4	1	$[3, 6]$

Table 2: Topological properties of the network scenarios.

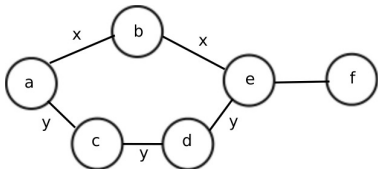

Figure 2: Toy case example (quickest vs shortest path)

turned on for a short while at node startup.

The primary (and secondary) path is first set if not existent yet. Also, if the newly overheard path is shorter than the primary path $|\mathcal{L}_{jd}| < |\mathcal{P}_{jd}|$, then the primary path is updated with the new candidate. Similarly, if the overheard path has lower similarity than the current secondary path $|\mathcal{P}_{jd} \cap \mathcal{L}_{jd}| < |\mathcal{P}_{jd} \cap \mathcal{S}_{jd}|$, or if it has equal similarity but is shorter than the secondary $|\mathcal{P}_{jd} \cap \mathcal{L}_{jd}| = |\mathcal{P}_{jd} \cap \mathcal{S}_{jd}| \wedge |\mathcal{L}_{jd}| < |\mathcal{S}_{jd}|$, then the secondary path is updated. Not shown in the pseudocode for the sake of simplicity, ties in the secondary path selection are broken at random when also $|\mathcal{L}_{jd}| = |\mathcal{S}_{jd}|$.

All nodes then add their own identifier and input/output ports to the frame m, the node probabilistically transmits the message with independent decisions per each interface (except the one from which the message has been received), and updates the per-source counter C_s.

4. Scenarios and methodology

In the following sections we thoroughly analyze the performance of APLASIA by means of analysis, Omnet++ simulationsand Click experiments. Simulations are run on several network topologies, whose main characteristics (such as size N, average δ and standard deviation σ_δ of node degree, average link propagation delay Δ and network diameter D) are reported in Tab. 2. While most of the networks correspond to real topologies (e.g., either L2 topologies as Geant and Tiger2, or L3 topologies inferred by Rocketfuel [24]), we also consider Erdos Reny random graphs of variable degree and size (up to 10,000 nodes), to stress test APLASIA performance (as done in [7]).

Our evaluation focuses on the discovery of a primary and a secondary path. We compare the quality of APLASIA paths with respect to the optimum gathered via the centralized solution in Viking[3], and that we described early in Sec. 2. We instead compare the control plane overhead against a classic distributed LS algorithm.

The remainder of the evaluation is organized as follows. Sec. 5 investigates by analytical modeling and simulation the APLASIA path quality vs cost tradeoff for varying β. We express path quality in terms of connectivity and optimality and path computation cost in terms of the number of transmitted messages. Then, Sec. 6 assesses APL system dynamics. We again resort to simulation to estimate the convergence time of the secondary path, and the duration of the whole advertisement process. We analytically model the time evolution of message propagation to better investigate its auto-termination property. Finally, we report on our ongoing effort in Click implementation in Sec. 7.

5. Path computation performance

In this section, we dissect several aspects of APLASIA path computation. To gather representative performance of APL, we simulate a single complete advertisement round (i.e., where each node performs a single advertisement), and exhaustively consider all the multiple computed paths interconnecting any two nodes pair in the network. For each parameter setting, results are averaged over 20 simulations runs (i.e., to smooth out different randomized advertisement orders and probabilistic decisions). Notice that we do not expect, in practice, all nodes to learn advertised paths: indeed, core devices performing only switching functionality will likely remain stateless. At the same time, this approach allows to assess quality of the primary and secondary paths that APLASIA is able to find *between any node pair*, thus allowing to get an unbiased picture of APL performance.

5.1. Primary and secondary path quality

We expect APL messages traveling on the *shortest path* to reach a node *before* messages that take longer paths: this definitively holds in case of homogeneous delay. Otherwise, it may happen that messages traveling along the *quickest path* arrive first, which could be then stored as primary path. As shown by the toy-case of Fig. 2, this can happen in networks having links with

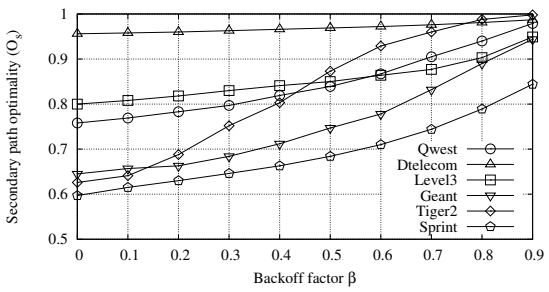

Figure 3: Optimality of secondary APL paths.

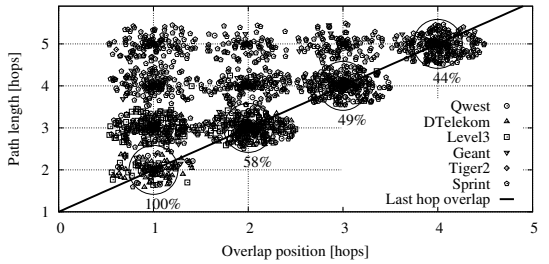

Figure 4: Scatter jittered plot of the secondary overlap position versus the length of the primary path.

very long delays. In Fig. 2, whenever the $2x > 3y$ condition on the link delay holds, an advertisement from a will reach node e on the path $a - c - d - e$ before $a - b - e$: hence, in this toy case, node f will receive the message having traveled over the shortest path from a with probability β.

In practice, simulation results testify excellent connectivity and quality properties of the *primary* path, as (i) APLASIA nodes are always connected irrespectively of β and of the network topology, (ii) when $\beta > 0$ the shortest path is found in more than 95% of the cases and (iii) the quickest path is selected as primary in the remaining cases.

Similarly, APLASIA nodes are always connected on more than one path. Hence, a more challenging goal is that of finding *optimal secondary* paths, whose quality depends on the backoff β. Fig. 3 reports the probability that the secondary path found by APL is optimal, for all topologies and varying β (we recall that $\beta = 0$ corresponds to the case where APL sends no additional messages w.r.t a classic LS routing algorithm). As Fig. 3 shows, higher values of β translate into higher quality of the secondary path (e.g., over 80% of the secondary paths are optimal for all networks except Sprint when $\beta = 0.7$), although in case of parameter misguidance (i.e., too low β) the path quality degrades gracefully.

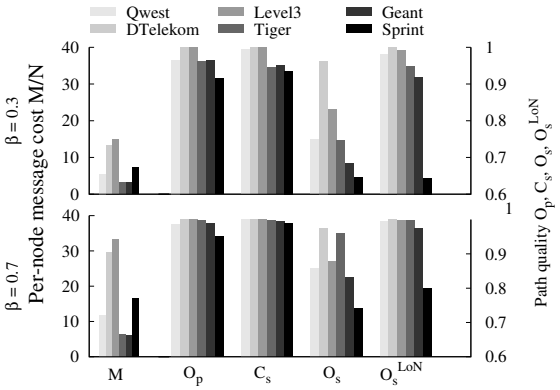

Figure 5: Refined performance with LoN heuristic.

5.2. *Refining APL*

We now pin-point the root cause of non-optimality and propose an effective counter-measure. As pointed out in [25], in most ASs failures happen nearby to the edge nodes (at most two hops far), and rarely internally to the AS topology: hence, paths should be as disjoint as possible n*ext to the edge origin and destination*. Motivated by [25], we cope with this issue close to the source of the advertisement, by letting the advertiser introduce PI information for the whole list of its neighbors (LoN) in the advertisement frame. LoN information is useful to nodes whose primary and secondary paths overlap next to the advertiser: as nodes may have gathered paths toward any of the LoN neighbors during previous advertisement rounds, they may decrease the overlap of the secondary path by selecting a more disjoint path through one of the advertiser neighbors.

We report in Fig. 4 the relevance of the above observation by plotting the position of the overlap in case LoN is *not* included in the advertisement frame. Position expresses the distance in hops from nodes receiving the advertisement to the advertisement originator, so that higher values correspond to overlap next to the advertiser. Fig. 4 reports a scatter plot of the overlap position versus the path length, using random jittering to enhance the visual presentation. Along the diagonal, we report the percentage of cases (over all topologies) where carrying LoN in the advertisement message could have helped in reducing the share of faith (i.e., the overlap) between paths.

Fig. 5 reports, at a glance, the quality of the multiple discovered paths along with the communication cost, for different networks and values of the backoff parameter β. Cost is reported on the left y-axis, expressed in terms of the number of messages M/N handled by each node during a single advertisement procedure (av-

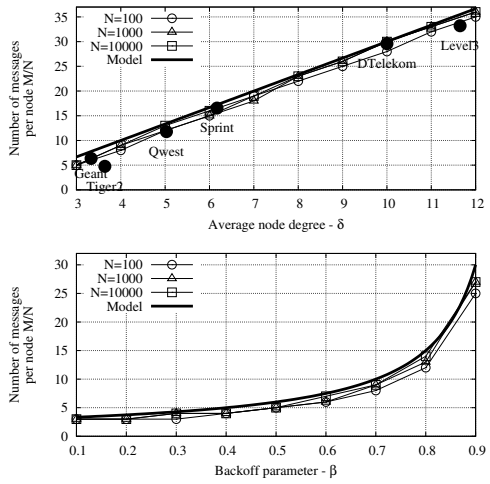

Figure 6: Model vs simulation comparison of the number of messages per node for varying node degree δ when $\beta = 0.7$ (top) and backoff parameter β when $\delta = 4$ (bottom).

eraged over all advertisements). Path quality metrics are instead reported on the right y-axis: namely, the probability to find optimal primary path O_P, the probability of being connected C_S on the secondary path, and the probability that the secondary is optimal respectively with (O_S^{LoN}) and without (O_S) the LoN heuristic (we avoid reporting C_P as nodes are always connected on the primary).

Notice that the raw number of messages is very limited for both values of β, so that we report $\beta = 0.3$ as an example of APLASIA performance under wrong parameter settings. Then, notice that the shortest path is discovered in most of the cases and that, especially for high β, a secondary path is always found. The quality of the secondary path (i.e., the fact that this secondary path is the shortest path most disjoint from the primary) is instead strongly affected by the LoN heuristic. Notice indeed that LoN affects optimality more than β, as can be easily gathered by comparing O_S^{LoN} vs O_S. Moreover, as O_S value without LoN is already high, the use of the LoN heuristic is able to cope with the remaining overlaps in Fig. 4, so that APLASIA is very often able to find optimal secondary paths O_S^{LoN} as well. More precisely, with LoN heuristic and $\beta = 0.7$, more than 80% of the secondary paths are optimal over all networks including Sprint (or more than 98% for all networks excluding Sprint).

5.3. Modeling the advertisement cost

The total number of control messages M transmitted over the network during a single advertisement process can be easily estimated neglecting the actual network topology. Considering for simplicity a time slotted execution of the APL algorithm, we have that if node s has started the advertisement process, a generic node j sends $\delta - 1$ advertisement messages (i.e., on every interface except for the interface from which the message came) with a probability (1) that depends on the number of times it previously handled the message from the same advertiser. The total network message count can then be gathered by simply summing up over all possible counter values $C_j \in [0, \infty]$, and taking into account that all nodes behave the same multiplying by N:

$$\begin{aligned} M &\approx N\left((\delta-1) + (\delta-1)\beta + (\delta-1)\beta^2 + \cdots\right) \\ &= N(\delta-1)\sum_{n=0}^{\infty}\beta^n = N\frac{\delta-1}{1-\beta} \end{aligned} \quad (2)$$

Notice that (2) is a conservative estimate of the number of messages, as we do not take into account that loops form and thus messages get discarded. A trivial refinement to (2) would be thus to truncate the sum to N, with a minimal impact as β^n becomes negligible for large n.

Above all, (2) shows that the number of messages M distributed over the network during an advertisement linearly depends on the network size and average degree δ, and hyperbolically on $1 - \beta$. While (N, δ) are given by the scenario and cannot be changed in actual deployment, the exponential backoff β gives a very simple knob to tune the overhead with respect to LS. The number of additional messages is off of at most an overhead factor $1/(1-\beta) > 1$, ensuring APL to be scalable as the overhead beyond LS is tunable and bounded.

We validate (2) against simulation in Fig. 6, where we normalize the number of messages over the network size to simplify the comparison over networks having heterogeneous sizes. We simulate both the real network topologies early considered (filled circles, to show model accuracy in practice) as well as random graphs with varying degree $\delta \in [3, 12]$ and size $N \in \{100, 1000, 10000\}$ (empty points, to stress test APL). Fig. 6 reports the number of messages handled per node M/N during an advertisement round, comparing model and simulation for varying network degrees (top) and β values (bottom), from which is easy to gather in both cases an excellent matching.

Aside the model accuracy, notice that the number of messages generated is very similar for $\beta \leq 0.3$: hence, the path quality results shown earlier for $\beta = 0.3$ can be considered as a lower bound for APLASIA performance. Intuitively, this happens due to the fact that messages are certainly flooded the first time, which already allows to discover more than the primary path. Any further mes-

sage transmission, for higher values of β, contributes to the refinement of the path quality. However, these refinements may come to the cost of an increase of M, so that alternative techniques (e.g., LoN heuristic) may be preferable than a mere increase of β.

6. Dynamic system performance

We next turn our attention to the temporal properties of the system, examining the duration of the path discovery process, and its auto-termination properties. Since the propagation time is the dominant delay component, we expect time-related properties to be affected by the geographical extension of the network, with possibly wide performance variation across scenarios of Tab. 2. Hence, we also develop a model to gather further intuitions on how time-related properties scale with the network size, without being bound to specific topological instances.

6.1. Path computation timeliness

First, we assess the path convergence time under APL, contrasting it with that of classic LS. As it is delicate to directly translate LS computational complexity in a temporal duration (as this depend on the processing speed of the network device), we prefer to resort to real-world performance of LS algorithms known in the literature [22] for a comparison. Furthermore, [22] analyzes a subset of the topologies we consider in this work (namely, Geant and a large ISP with topology size slightly smaller than Sprint), so that their findings are relevant for our analysis.

As [22] points out, the main source of delay in the convergence of OSPF and ISIS is (i) the FIB update process, with times on the order of *hundreds* of milliseconds, that depend on the processor speed (e.g., GRP, PRP1, PRP2, etc.). The second source is the (ii) execution of shortest path algorithm that requires *tens* of milliseconds at the highest processing speed on large networks (while, clearly, more sophisticated multi-path algorithms such as those used in BANANAS[5] or, worse, Viking[3], could significantly raise the impact of algorithm execution on the convergence time). The third source is (iii) the duration of the flooding process.

We point out that in APLASIA steps (i) and (ii) are skipped altogether and that multiple paths are already available *during* the flooding process, with thus a significant gain in the convergence speed. To evaluate the timeliness of the path computation process we now fix $\beta = 0.7$ as a good compromise between path optimality and extra message cost. Since the primary path is quickly established in APL, path computation converges when a node no longer updates its secondary path (with respect to the algorithm pseudocode, this time correspond to the last execution of line 13). As before, to gather unbiased performance, we let each node start an advertisement, repeating simulation 20 times to average the probabilistic decisions. During each round, we measure the time elapsed between the start of the advertisement at the source and the secondary path convergence at all nodes.

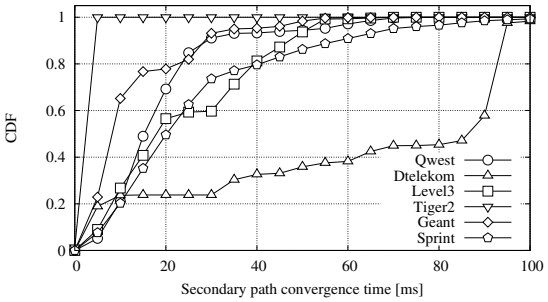

Figure 7: CDF of the time employed to converge on the selection of the secondary path (i.e., last execution of line 13 of APL).

The cumulative distribution function (CDF) of the path convergence time is reported in Fig. 7 for all networks. As expected, the raw duration is mostly affected by the average link delay: for all networks except DTelekom (for which the average link delay $\Delta = 17.2$ ms is much larger than that of the other topologies, cfr. Tab. 2), 90% of the secondary paths converged in less than 60 ms, and in any case convergence takes less than 100 ms in the *worst case* (while in [22], LS convergence takes more than 100 ms in the *best case*). This confirms the soundness of APLASIA design, that provides fast convergence of multiple paths.

6.2. Advertisement auto-termination

We now investigate the temporal evolution of the message propagation during a single APL cycle. With respect to Sec. 6.1, we take a complementary view and focus on the duration of the flooding process over the whole network, i.e., before the exponential backoff let the flooding die out.

The objective is not to accurately predict the number of sent messages, which is in any case bounded by (2), but to estimate some critical temporal properties, as for instance the time at which the flooding process reaches a peak, the time at which it vanishes, the impact of the network size, etc. We consider again a single advertiser and assume homogeneous propagation delays so

that time can be considered as slotted. For all $n \geq 0$, the mean number of messages $m(n)$ sent by the N nodes in the network at round n satisfies the approximate recursive equation:

$$m(n+1) \approx m(n)\delta \sum_{k=0}^{\infty} P(k,n)\beta^k, \qquad (3)$$

where $P(k,n)$ is the probability that a node has received k messages until round n. By convention, the advertiser starts the flooding at round 0 so that $m(0) = 1$.

The approximation in (3) is valid for large graphs, say $N \geq 100$. For smaller graphs, errors in (3) comes both from (i) loops that stop the flooding process and from (ii) the number of edges on which each message is transmitted, which is approximated by δ. Since a message arriving at some node is not sent on the corresponding incoming interface in the actual system, the number of edges on which the message is broadcasted has a binomial distribution with parameters $N-1, \delta/N$: the corresponding mean is $(N-1)\delta/N$, which is close to δ for large N. With respect to our previous notation, we have that $\sum_{i=0}^{\infty} m(i) \approx M$: while M counts the number of messages sent on output interfaces, $m(n)$ counts all messages in flight at a given round (i.e., including the one received from the interface where APL avoids flooding).

Still, the probability distribution $P(k,n)$ in (3) needs to be estimated. We model $P(k,n)$ by noticing that each node has handled $\bar{M}(n) = \sum_{i=0}^{n} m(i)/N$ messages on average up to round n. Since a large number of messages are sent, the distribution of the number of received message is approximately Poisson of parameter $\bar{M}(n)$ at each node, i.e., $P(k,n) \approx e^{-\bar{M}(n)}\bar{M}(n)^k/k!$. Using (3), we deduce:

$$m(n+1) \approx m(n)\delta e^{-(1-\beta)\bar{M}(n)} \qquad (4)$$

that we can compute numerically. The quality of the approximation is illustrated in Fig. 8, showing comparison of the numerical solutions of (4) with simulation of random networks, varying the network size up to $N = 10000$ at fixed backoff $\beta = 0.7$ and degree $\delta = 4$ for the sake of illustration (similar matches are gathered for any other parameter settings). For convenience, we normalize the number of messages over the network size $m(n)/N$, to ease the comparison over heterogeneous network sizes.

The model closely captures the bell shape of the message propagation, which confirms that the Poisson assumption holds well in practice. In more details, as shown in Fig. 8, message dynamics reflect an initial

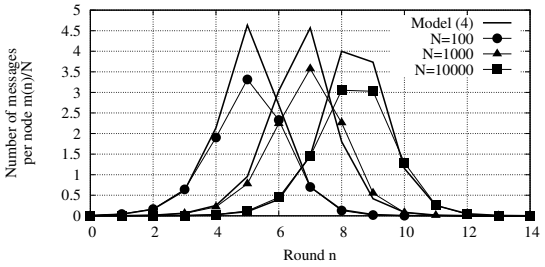

Figure 8: Message dynamics during an advertisement.

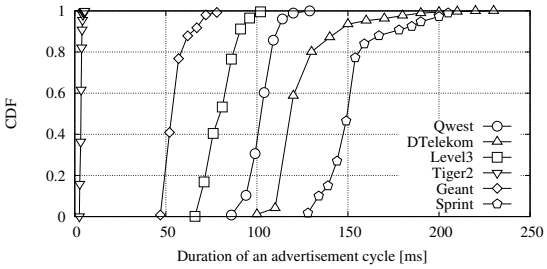

Figure 9: Duration of an advertisement cycle.

exponential rise due to flooding (as counters are initially 0, hence certainly transmitted since $\beta^0 = 1$). As soon as frame duplicates are received over the network, the exponential backoff kicks in and slows down the message increase, until a peak is reached (at round $n_{peak} = \text{argmin}_n m(n) \geq m(n+1) > 0$), after which the number of messages progressively decreases (and completely stops at about $2n_{peak}$).

From the picture, we gather that n_{peak} increases (i) logarithmically with the network size N, or (ii) linearly with the graph diameter (since in random graphs the diameter scales logarithmically with the network size [26]). This is intuitive, since the flooding slows down as soon as a node starts receiving multiple copies of the message, which is always the case when the path length reaches the network diameter.

6.3. Duration of an advertisement cycle

While (4) is useful to show the auto-termination property of APL, it is however not helpful in determining the duration of the process in real networks, for which we resort to simulation. Fig. 9 reports the CDF of the duration of a single APL advertisement, averaging over 20 repetitions. Comparing Fig. 9 against Fig. 7, we gather that secondary path usually converges earlier with respect to the whole advertisement duration. Moreover, the advertisement still remains short, as it never exceeds

250 ms for the real topologies of Fig. 9 (nor it would in a $N = 10000$ nodes random graph with 10 ms links delay as for Fig. 8, since the number of in-flight messages goes to 0 before the 15-th round).

7. Click implementation

We have implemented the core APLASIA functionalities in a Click modular router. Our modules fully implement the data plane frame processing and maintenance capabilities, but only partly implement APL control plane functionalities for the time being. To give a rough idea of the implementation complexity, APLASIA Click modules account for about 5000 lines of code, 24 classes and 65 functions.

To summarize the main functional blocks, each physical interface is connected to an APLASIA Port (AP) module, consisting of two distinct Edge Origin (EO) and Edge Destination (ED) sub-blocks. These blocks handle the communication to and from the physical interface, the encapsulation/decapsulation of data from/to the upper layer, the generation of path discovery request/reply, etc. To each AP module is associated an APL module, that assists the AP by duplicating discovery frames to the other ports for path computation. All AP modules of a node are interconnected through a single APLASIA Matrix (AM), whose main aim is to perform the stateless switching function (by inspection of the APLASIA autoforwarding header) and that holds node-wide state (such as the path cache and APL counters).

Primarily, we used the Click implementation for functional verification of the APLASIA principles. In this section, however, we report on preliminary experimental results gathered in a small size testbed, that we use to assess the limits and capabilities of our current implementation.

The testbed is composed by seven PCs equipped with dual-core Intel Xeon E3110 CPUs, clocked at 3.00GHz, equipped with 4 line cards (i.e., four AP and APL blocks each). PCs are arranged as a bus topology, and are interconnected by two point-to-point 100 Mbps Ethernet links[10]. In our setup, the origin node runs in user space (so that it is easy to access timestamping functions without modifying the Click code), while all the others nodes run in kernel mode[11].

We devise two simple yet instructive test scenarios to benchmark the Click implementation, aiming at gathering the atomic duration of (i) data plane forwarding function t_{fw} and (ii) handling of control plane frames during the discovery process t_{cp}. In order for the benchmark to be the least intrusive as possible, we avoid to explicit instrumenting the Click code (e.g., by means of timestamping and `click_chatter` calls). Moreover, we avoid relying on time synchronization, as the precision needed to gather reliable figures exceeds NTP capabilities. Therefore, we decide to infer the atomic cost of message handling in the data/control plane from Round Trip Time (RTT) measurements at different path lengths h. We devise two simple models that account for the data $RTT_{dp}(h)$ and control planes $RTT_{cp}(h)$ respectively, that we later fit on experimental data from our testbed:

$$RTT_{dp}(h) = 2t_g + 2h(t_{tx} + t_{fw}) \quad (5)$$

$$RTT_{cp}(h) = 2t_g + 2h(t_{tx} + t_{fw} + t_{cp}) \quad (6)$$

In the above expressions, h represents the path length, t_g is unknown value accounting for the message generation time in the AP functional block (counted twice due to the response message), t_{tx} accounts for the message transmission time over the line card (which is known and given by the ratio of the actual frame size over the link capacity). The duration of the stateless switching operation performed by the AM block on a single data frame is represented by t_{fw} (i.e., header inspection and message passing to the right output port), while t_{cp} accounts for messages handling in AP.

We start by determining t_g and t_{fw} by fitting (5) with a first set of experiments, after paths have been established. We carry on experiments for varying path lengths $h \in [2, 6]$, generating 200 RTT samples per h value. In order to measure the RTT, we use an APLASIA OAM tool behaving as the classic ICMP `ping` command of IP networks (i.e., automatically generating response messages). OAM messages have known size, which is accounted for in t_{tx}.

Experimental results are reported in Fig. 10, that depicts, for each hop length, the PDF of $RTT_{dp}(h)$ (shaded curves in the left plot) and the first 50 samples gathered during the experiment (points in the right plot). Both plots also show, with a solid black line, the fitting of (5) gathered using an implementation of the non-

[10] In the testbed, we merely use Ethernet cards as point-to-point transceivers between any couple of routers: in other words, no switching, learning or other Ethernet functionalities are used. As a side effect, encapsulation of APLASIA in an Ethernet frame testifies that APLASIA principles are agnostic to the underlying layer.

[11] We point out that, as user-space process can be pre-empted at kernel level, we gather conservative results that may slightly underestimate the actual APLASIA performance.

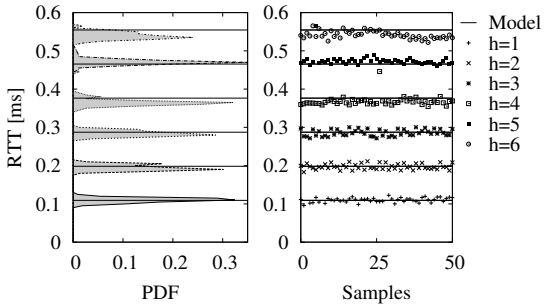

Figure 10: Fit of computational cost of data-plane forwarding operation in the Click implementation.

linear least-squares (NLLS) Marquardt-Levenberg algorithm. The fit converges to $t_g = 54\,\mu$s and forwarding of $t_{fw} = 45\,\mu$s, with very small asymptotic errors of 0.63% and 0.33% respectively: notice indeed the good match between the model and experimental data (despite a few outliers, possibly due to the user-space device).

We finally perform path advertisement experiments (in a slightly modified setup not detailed for lack of space), to gather t_{cp} by fitting (6) on further experimental data. The fitting yields an estimate for the control plane overhead of $t_{cp} = 48\,\mu$s (with an asymptotic error of 0.2%), which is thus of the same order of magnitude of the forwarding operation in the data plane. These results further highlight the practical interest of APLASIA and the lightweight of the control plane advertisement task.

8. Conclusions

We propose APLASIA, a new, flexible network architecture, requiring simple nodes hardware and little or no configuration. Stateless operation in the data plane is achieved by trading switching tables for header space, as frames carry a fully specified source-routed sequence of output ports identifiers in the frame header. This choice has two important consequences: first, APLASIA frames are autoforwarding, so that core devices need not to be equipped with (nor to maintain) FIB entries. Second, it becomes trivial to ensure that paths are loop-free.

APLASIA supports discovery of multiple disjoint paths through a simple yet effective distributed algorithm, able to discover nearly optimal path at bounded computational and communication complexities. The number of messages exchanged in the control plane remains of the same order of magnitude of classical link-state algorithms, i.e., $O(N\delta)$, as the use of probabilistic exponential backoff limits the overhead to a fixed multiplicative factor $\frac{1}{1-\beta}$. The algorithm requires simple comparison operations at each packet reception, so that the computational complexity is of the same order of magnitude when the set of paths is limited to $k = 2$.

We investigate the main APLASIA performance by means of extensive simulation (on real topologies and synthetic ones up to 10,000 nodes), analytical modeling of main systems aspects, and testbed experiments of our ongoing Click implementation. Our results highlight several desirable properties (such as auto-termination, rapid path creation process, near-optimal quality of the primary and secondary paths, graceful degradation in case of wrong parameter settings, etc.), that testify the overall interest for APLASIA.

While this work testifies the overall interest of APLASIA, it also leaves some open points, that we discuss in the following.

Larger path sets. While it may be computationally unfeasible to find the *best path set* of size $k > 2$ as [11] points out, it is however possible to simply find a set of k paths. Let consider that the algorithm has just found the primary path (often, the shortest is among the first to be selected). On the reception of a new message, the new path will be stored as the secondary (irrespectively of the path quality, since a secondary is not existent yet). On a further message reception, the latest received path will either be elected secondary (and shifts the secondary in the third position according to some criterion, which is possibly more complex than merely requiring disjointness from the primary path), or will be appended at the end of the list. Later, when the set of k paths is full, a replacement is possibly needed, and if the decision does not involve comparison among multiple paths, then a greedy algorithm would need to perform (at most) k comparison to take this decision. Hence, the computational complexity of a k-path extensions would remain tolerable $O(N\delta k)$: at the same time, while the quality of the resulting set may be good enough in practice, we acknowledge that this point requires a careful investigation.

Administrative routing weights. Another open point, that partly goes against the holistic, plug-and-play nature of APLASIA, concern the ability of "emulating" administrative routing weights. In our evaluation, APLASIA treats all link as having equal weight $w_i = 1, \forall i$: yet, we acknowledge that this choice may clash with the current ISP practice of employing administrative weights to bias the construction of the overall topology by specifying arbitrary costs for given links.

An interesting question that remains open is whether it would be possible to tweak administrative weights by using heterogeneous β_i values for different links i (let aside the problem of configuring the individual devices with heterogeneous β_i), and how to map w_i into β_i.

Failure resolution. While APLASIA finds multiple paths, it does not specify how failure resolution should be handled. In case route cache is stored only at the edge of the network, then in the worst case (i.e., when failure happens near the destination), the recovery time would be on the order of 1.5 RTT (i.e. one RTT to notify the source, plus the one-way delay for retransmission along the backup path).

More efficient failure resolutions could happen in case that (at least some) core devices would be equipped with a path cache: in this case, the message could backtrack toward the source until a node equipped with a path cache is found, that could relay the message on the alternate path (other possible techniques are surveyed in [27]). As [23] points out, by letting traffic to switch between paths at intermediate hops, a source gains access to as many as k^l paths to a destination, with k the number of slices and l the number of switching point on the path.

Notice that the presence of a path cache on a core device would *not* affect stateless autoforwarding in the data-plane under normal operation. At the same time, as now some of the core devices need to be equipped with a FIB, interesting questions would be thus: (i) to explore the tradeoff between the increased capital expenditure versus the improvement on failure handling and (ii) the optimal placement of switching points in the topology.

Amount of control plane state To simplify the description, we assumed that each node keeps a set of $O(N)$ counters to take its probabilistic decision. While this amount of state is scalable with respect to the $O(N^2)$ state required by [21], however it is possible to further reduce the state space requirements. We point out that the main issue is not on the raw amount of state, which is very limited as counters have byte-size (as for practical purposes β^{255} can be considered 0), but to ensure scalability and make a more efficient use of *all* hardware resources, in spirit with APLASIA (and Ockham's razor).

In the normal mode of operation, it will be unlikely for *all* nodes to contemporary start advertisement cycles, though it would be desirable to tolerate some contemporary advertisements. From the evaluation, we know that advertisement has a short duration, moreover slowly growing with the network size. In normal operation it could be thus easy to desynchronizing the start of advertisement operations, by simply employing techniques similar to the Carrier Sense Multiple Access (CSMA) based approach employed in the original IEEE 802.3 Ethernet LANs.

Without entering in details, we have preliminarily tested a CSMA-based approach, requiring that nodes sense whether there is some advertisement ongoing prior to start a new one: in case they receive some advertisement messages during a carrier sense interval, they backoff (as in p-persistent CSMA), otherwise they start the advertisement. The mechanism is imperfect: when other advertisements already have started in some faraway region of the network, but no message attains a node while it performs carrier sense, the latter can start a new advertisement in turn. However, as opposite to destructive interference in medium access protocols, the start of contemporary advertisement do not lead to severe issue provided that the *ongoing* advertisement are counted on a independent counters. In practice, since the whole duration of the advertisement cycle is subsecond, is easy to bound the number of concurrent advertiser (e.g., less than ten with very high probability) by choosing a carrier sense interval on the order of the duration of an advertisement (e.g., on the order of hundreds of milliseconds) – leading to a tolerable delay in normal operation.

As counters need to be reset at each new advertisement cycle, we can implement the counter set as a FIFO queue: when a new control message is overheard, the oldest counter is pushed out of the FIFO, and a new one is inserted and reset. Note that this approach would not only reduces the amount of state to $O(1)$, but could also further contribute in making APLASIA plug-and-play as the good'ol Ethernet. This direction is also part of our ongoing work on Aplasia.

Acknowledgements

The work presented in this paper has been carried out at LINCS (`http://www.lincs.fr`) and was funded by an Orange Labs grant. We are thankful to Thomas Bonald for the initial fruitful discussions.

References

[1] T. L. Rodeheffer, C. A. Thekkath, D. C. Anderson, Smartbridge: a scalable bridge architecture, in: ACM SIGCOMM, 2000.

[2] R. Perlman, Rbridges: transparent routing, in: IEEE INFOCOM, 2004, pp. 1211 – 1218.

[3] S. Sharma, K. Gopalan, S. Nanda, T. Chiueh, Viking: a multi-spanning-tree ethernet architecture for metropolitan area and cluster networks, in: IEEE INFOCOM, 2004.
[4] C. Kim, M. Caesar, J. Rexford, Floodless in SEATTLE: a scalable Ethernet architecture for large enterprises, in: ACM SIGCOMM, 2008, pp. 3–14.
[5] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, A. Gandhi, Bananas: an evolutionary framework for explicit and multipath routing in the internet, in: ACM FDNA, 2003, pp. 277–288.
[6] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, ROFL: routing on flat labels, in: ACM SIGCOMM, 2006, pp. 363–374.
[7] A. Singla, P. B. Godfrey, K. Fall, S. Iannaccone, G. Ratnasamy, Scalable routing on flat names, in: ACM CoNEXT, 2010.
[8] C. Betoule, T. Bonald, R. Clavier, D. Rossi, G. Rossini, G. Thouenon, Adaptive probabilistic flooding for multi-path routing, in: IFIP NTMS, 2012.
[9] N. R. Mysore, A. Pamboris, N. Farrington, P. Huang, N. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in: ACM SIGCOMM, 2009.
[10] D. Eppstein, Finding the k shortest paths, IEEE FOCS (1994) 154–165.
[11] J. Mudigonda, P. Yalagandula, M. Al-Fares, J. Mogul, Spain: Cots data-center ethernet for multipathing over arbitrary topologies, in: USENIX NSDI, 2010, pp. 18–34.
[12] R. Sanchez, L. Raptis, K. Vaxevanakis, Ethernet as a carrier grade technology: developments and innovations, IEEE Communications Magazine 46 (2008) 88–94.
[13] Ieee standard 802.1ah - provider backbone bridges (pbb) (June 2008).
[14] Ieee 802.1qay - provider backbone bridge traffic engineering (pbb-te) (Jan. 2009).
[15] E. Mannie, Generalized multi-protocol label switching (gmpls) architecture, in: IETF RFC3945,.
[16] D. Johnson, Y. Hu, D. Maltz, The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4, IETF RFC 4728 (Feb. 2007).
URL http://www.ietf.org/rfc/rfc4728.txt
[17] J. J. Garcia-Lunes-Aceves, Loop-free routing using diffusing computations, IEEE/ACM Trans. Netw. 1.
[18] F. Solano, T. Stidsen, R. Fabregat, J. Marzo, Label space reduction in mpls networks: How much can a single stacked label do?, IEEE/ACM Transactions on Networking, 16 (6).
[19] P. B. Godfrey, I. Ganichev, S. Shenker, I. Stoica, Pathlet routing, in: ACM SIGCOMM, 2009, pp. 111–122.
[20] W. Zaumen, J. Garcia-Luna-Aceves, Loop-free multipath routing using generalized diffusing computations, in: IEEE INFOCOM, 1998, pp. 1408–1417.
[21] M. J. Blesa, C. Blum, Ant colony optimization for the maximum edge-disjoint paths problem, in: Applications of Evolutionary Computing, Vol. 3005, 2004, pp. 160,169.
[22] P. Francois, C. Filsfils, J. Evans, O. Bonaventure, Achieving sub-second igp convergence in large ip networks, ACM SIGCOMM CCR 35 (2005) 35–44.
[23] M. Motiwala, M. Elmore, N. Feamster, S. Vempala, Path splicing, in: ACM SIGCOMM, 2008, pp. 27–38.
[24] N. Spring, R. Mahajan, D. Wetherall, Measuring isp topologies with rocketfuel, in: ACM SIGCOMM, 2002, pp. 133–145.
[25] N. Feamster, D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, Measuring the effects of internet path faults on reactive routing, ACM SIGMETRICS (2003) 126–137.
[26] B. Bollobas, Random Graphs, Cambridge University Press, 2001.
[27] A. Raj, O. C. Ibe, A survey of ip and multiprotocol label switching fast reroute schemes, Computer Networks 51.