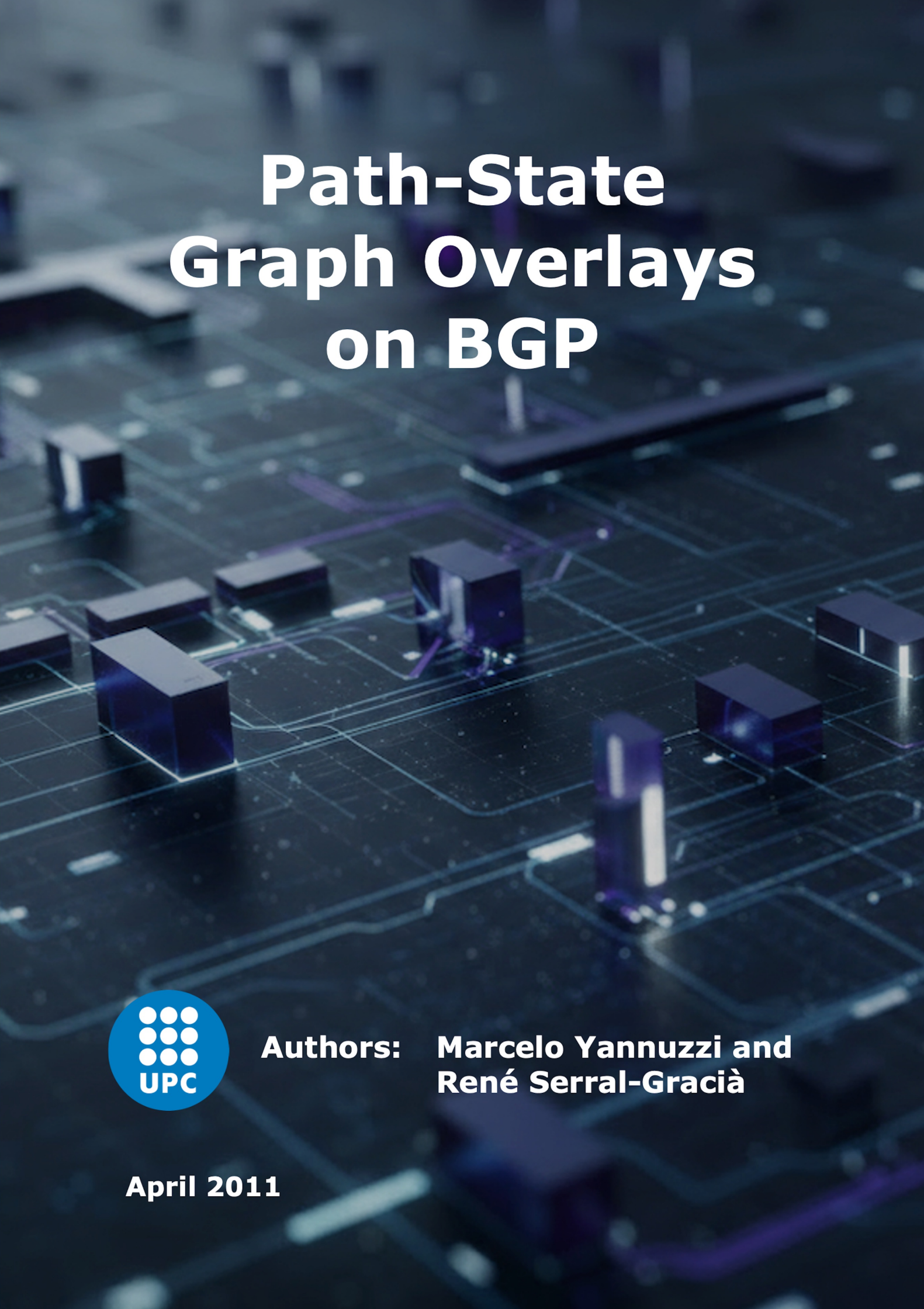


Path-State Graph Overlays on BGP

Authors: Marcelo Yannuzzi and René Serral-Gracià

April 2011

Abstract

This document describes a new control protocol for the interdomain routing system called *Path-State Protocol* (*PSP*), and discusses its implementation in the form of an overlay over the Border Gateway Protocol (BGP). This overlay is designed to coexist with BGP, and its main task is to help improving the performance and security of the latter in a non-disruptive way. The key to achieving this goal is that the overlay protocol allows domains to construct and maintain a link state-like graph of the Internet, which we call a *Path-State Graph* (*PSG*). The PSGs capture the interconnections between Autonomous Systems (ASs) while respecting the opaqueness of providers, i.e., ISPs can hide their interdomain connectivity in the PSGs according to the usual routing policies between domains. The overlay of PSGs over the BGP protocol provides a straightforward way of combining the strengths of link-state and path vector routing, without requiring the replacement of the BGP protocol. This approach can help improving different aspects of the interdomain routing system, such as speeding up the routing convergence, reducing the churn of routing updates, as well as performing path and origin inspection, and validation, in a pragmatic way.

We discuss the design and specification of the building blocks needed to implement the PSP/BGP protocol interactions, including, the Application Programming Interface (API) enabling the communication between BGP and the PSP overlay, and the methods for configuring peers in the overlay layer. This document also specifies the messages and the strategy used for updating the overlay control information in a compact routing optimal way. Regarding the implementation, we describe the integration with a Quagga BGP router, though special care has been taken to ease the migration to other router operating systems, such as Cisco IOS.

It is important highlighting that only minor modifications have been made to BGP for supporting the communication and coordination between the latter and the upper control layer. Indeed, all the functions of a BGP speaking system remain unmodified, except for the way in which BGP advertisements are controlled and processed. Events that involve either the generation of BGP updates, or the processing of updates received from a BGP neighbor, are inspected by the top layer, and, depending on the event, the updates might follow the normal BGP flow or be entirely processed by the top layer. The latter means that the overlay can take control, process the event, and, if needed, use the PSP control protocol for updating the entire interdomain routing system without triggering a single BGP update—indeed, we shall show that the inspection and processing performed by the overlay layer is carried out transparently to BGP.

The focus of this SRA is on prototyping the overlay protocol and its interaction with BGP for testing of the convergence and churn properties, but it is important highlighting that the design was devised with the inspection of routes and their origins right from the start, rather than as an add-on. These features shall be developed in a second stage.

Contents

1 Introduction

The motivations for the integration of link-state and path vector routing are well known [1], and can be summarized as follows. Link-state protocols offer fast convergence, low churn rate of route advertisements, quite effective Traffic Engineering (TE) tools, and also facilitate the development of solutions for securing the routing. Path-vector protocols, on the other hand, can handle routing policies, ease loop detection, and highly improve the scalability of the routing system. Some initiatives have tried to achieve this integration in the past, but without practical success (see, e.g., HLP [2]).

In this document, we present a new and pragmatic approach, where emphasis is put on the lessons learned from previous attempts for integrating link-state and path vector routing. In particular, in the need to furnish solutions that do not require the replacement of the Border Gateway Protocol (BGP), and the requirement to respect the privacy of provider's peering and routing policies while constructing link state-like graphs of the network. In this framework, this document specifies a new control protocol for the Internet called *Path-State Protocol* (*PSP*), and discusses its implementation in the form of an overlay over BGP, where our goal is:

- to keep the distribution of IP prefix reachability and the computation of loop-free paths in the scope of BGP—the underlay in Figure 1;
- and to overlay the management of new control plane tasks in domain-level functions. These functions include mechanisms for constructing, advertising, and maintaining policy-based graphs of the AS-level interconnectivity of the network, which we call *Path-State Graphs* (*PSGs*) (cf. Appendix A), as well as mechanisms for inspecting and controlling routing updates, and thereby avoid the churn and poor performance of BGP during a routing convergence (cf. Figure 1). These overlay functions are conceived to assist BGP and they are supported by the PSP protocol.

In summary, this document specifies how an overlay of control functions can be introduced in a non-disruptive way, and how these functions can be used to seamlessly interact with BGP. Clearly, this approach facilitates the evolution of the routing system, since new control functions can be progressively integrated into the upper layer without affecting the BGP code.

In the remainder of this document, we will first describe the general operation and the interaction between the PSP and BGP protocols. Then, we will focus on the internals of a particular router Operating System (OS), namely, Quagga [3], and explain the minor changes needed to integrate the desired functionality into an already working BGP implementation. We have made strong efforts to avoid building on Quagga's implementation-specific details, and thereby minimize the overhead of migrating the solution to other router OSs, such as Cisco IOS. After that, we specify the set of operations of the overlay layer and its requirements, as well as the details in order to implement the PSP/BGP protocol suite. In particular, we provide specifications for the different protocols shown in Figure 1. Next, we analyze a set of use cases covering possible events and interactions between the two layers, with the aim of improving different aspects of the BGP convergence.

The details regarding the theory and computation of Path-State Graphs (PSGs) can be found in Appendix A.

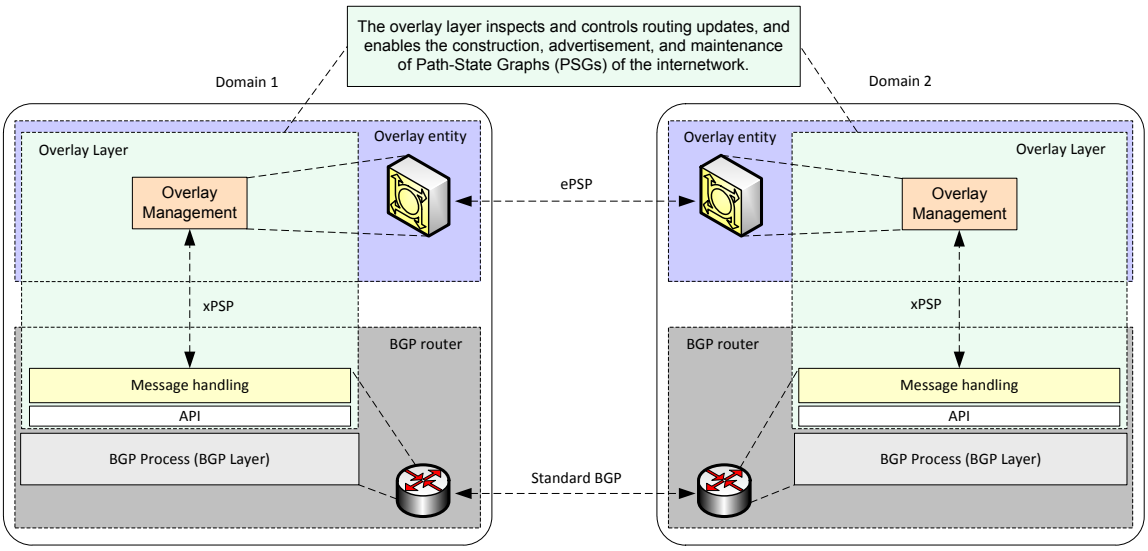


Figure 1: General view of the architecture and the interactions between layers. Note that the bottom layer is provided by the BGP protocol, whereas the top layer is provided by the Path-State Protocol (PSP)—ePSP stands for external PSP and xPSP stands for cross-layer PSP.

2 Summary of Operation of the Protocol Suite

Figure 1 depicts the different parts that compose the PSP/BGP protocol suite. As mentioned above, the interactions and coupling between these protocols aim at a seamless and non-disruptive integration with the existing routing system. Two "entities" can be clearly identified: 1) the BGP routers and 2) the *Overlay Entities* (*OEs*) which are the ones in charge of constructing, exchanging, and maintaining the PSGs. The OEs are also the ones that perform the inspection of BGP messages, and, when necessary, they can take control over BGP regarding the generation and advertisement of routing updates. The OEs can be implemented either using dedicated (external) hardware or be embedded into the router itself. The case shown in Figure 1 is the one using dedicated hardware. In that case, the overlay layer is split into two functional blocks, one on the router OS side and the other on the OE side—note that the implementation of the "overlay part" on the router side is not part of the BGP process; it is a separate and independent process running on the router's OS.

The communication between these two processes within the router's OS is managed through a new Application Programming Interface (API). The advantage of this strategy is threefold. First, it minimizes the changes required on BGP's Finite State Machines (FSMs) [4]—only minor changes are needed on the BGP code. Second, the API provides a neat and simple way for handling the inspection and control of BGP advertisements through the overlay. Indeed, the implementation of the overlay part required on the router side is very simple, since it basically needs to manage the interactions with the BGP message queues, and the calls for modifying the BGP Routing Information Base (RIB) and/or the Forwarding Information Base (FIB) when needed. Third, this strategy facilitates the testing of the implementation with other router OSs (e.g., with Cisco IOS) since the changes on BGP and the router OS are minimized. Indeed, the software development effort is actually concentrated on the OEs, which,

as shown in Figure 1, can be implemented outside the router OS, thus making easier the migration tasks and the initial testing with other router OSs. Notwithstanding, the design does not exclude the possibility of integrating the implementation entirely on the routers.

In addition, Figure 1 shows that the communication between the overlay part on the router side and its counterpart on the OE side is handled through the PSP protocol. It is worth highlighting that when the OEs are implemented using external hardware, the design foresees that each OE may manage more than one BGP process in the router, or just a single one. The PSP connections between the OEs and the routers are referred to as *cross-layer PSP* (xPSP); the ones between OEs in different autonomous systems are referred to as *external PSP* (ePSP), whereas the PSP connections between OEs in the same autonomous system are referred to as *internal PSP* (iPSP)—the naming used is inherited from the separation between iBGP and eBGP in the BGP protocol.

An important aspect is that: *the information managed and distributed through the overlay layer is only used for control purposes. In other words, the data plane remains unmodified, and therefore, no specifications are required for the data plane in this document*.

2.1 The Role of the BGP Layer

The primary functions of the BGP layer are: i) to exchange IP prefix reachability information, and ii) to compute and distribute loop-free paths among domains. These functions are exactly the same that BGP systems perform today, so the role of BGP does not change. The only difference is that now the processing and advertisement of BGP messages is inspected, and, if necessary, controlled by the top layer. In particular, when a router receives a BGP advertisement from a neighbor, the message is enqueued in an overlay processing queue for inspection by the OE—this process will be described in detail in Section 3.2. The outcome of the inspection could range from a simple checking and resuming of the natural flow of BGP advertisements, up to the case where the OE takes complete control of the processing and further advertisement of the routing event. In the latter case, the overlay could instruct BGP that the RIB and/or FIB must be updated, and could even return control to that particular BGP message processing thread indicating that no further actions are required (meaning that no messages will be spawned from the router).

It is important to highlight that the update of the BGP RIB and/or FIB is always performed by BGP. Indeed, the changes in the routing and forwarding tables are committed exactly in the same way as BGP does today. The only variation is that events that might require changes on these tables are sequentially inspected and supervised by the overlay, so the latter only needs to make available the change/update for BGP on its corresponding queue. Note that concurrency and consistency issues are avoided in this way, since it is BGP the one that modifies the tables.

These interactions are enabled through the API between the BGP process and the overlay process embedded on the router OS (see Figure 1). The same processing is applied when, instead of receiving an update from a neighbor, a local event occurs that requires triggering BGP updates from the router (e.g., a link directly connected to the router fails, a local interface goes down, etc.).

In summary, the role of the BGP layer is the same as today. The only changes introduced are in the way in which BGP advertisements are controlled and processed, with the aim to improve the convergence, reduce the churn, and enhance the security features of the routing system.

2.2 The Role of the Overlay Layer

The PSP protocol embodies the whole set of operations and functionality provided by the overlay control layer. The implementation of this layer is distributed in two functional blocks, one on the router OS side and the other on the OE side—though as mentioned above, both can be fully integrated in the router OS. The role of the overlay control layer is twofold. In the first place, two OEs that belong to different autonomous systems will connect through the ePSP protocol to exchange "augmented" path vector information (see Figure 1). The term "augmented" means that each OE will export to its OE neighbors all the path vectors compatible with the routing policies in its local domain. Note that this is different from BGP-based routing, since each BGP router will only export its best route for any destination. At this point it is important to make two observations. Firstly, the granularity of the routing destinations managed within this "augmented" information is kept at the AS level, meaning that the OEs do not need to manage IP prefixes to build AS-level graphs of interconnectivity—this issue will be further elaborated and discussed in Section 2.2.2. Secondly, the exchange of this augmented information is entirely handled by the OEs, so BGP does not participate in any way in this process, and thus, no changes are required on the BGP code. The advantage of this approach is that it allows each OE to individually construct a graph of the AS-level topology (a Path-State Graph), through which the paths admitted by the routing policies can be consistently inferred (cf. Appendix A). More precisely, the ePSP protocol supports the distribution of policy-shaped graphs, thereby keeping the autonomy and expected isolation between routing domains. As detailed in Appendix A, these graph constructions will differ between domains, since this is the only way to ensure that transit domains can keep private their connectivity and routing policies to other domains (e.g., to their competitors). Appendix A also shows that despite these different topological views, the PSGs solve the inconsistencies in the inference of AS-paths on graphs.

The second role of the overlay layer is to exploit the graphs constructed to: a) dramatically reduce both the BGP convergence time and churn rate of BGP updates; and b) perform inspection and validation of routes in a realistic and pragmatic way. The enabler for these applications is the connection between the router OS and the OE through the xPSP part of the PSP protocol (see Figure 1). As mentioned before, through the xPSP protocol the overlay may take control and mandate the update of the RIB and/or FIB of the BGP router. With this approach, the overlay may instruct BGP to delete (insert) routes from (to) the router's RIB/FIB, for instance, to speed up the routing convergence—this particular case is further detailed in Section 3. It is worth noting that Bonaventure et al. [5] have already analyzed and discussed mechanisms for updating entries directly in the FIB of a high-end router.

The interactions and coordination between the overlay and underlay BGP control functions are analyzed in detail in Section 3 and the subsequent sections.

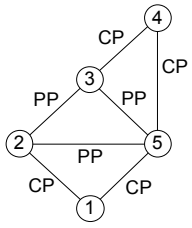
2.2.1 Advertisement and Storage of Path-State Graphs (PSGs)

As explained in Appendix A, the data structure that needs to be managed by the OEs for storing and maintaining the PSGs is very simple. To illustrate this, consider the example shown in Figure 2. Figure 2(a) depicts an AS graph and the commercial relationships between domains, while Figures 2(b) and 2(c) show the PSG constructed by an OE in AS1, and the Path-State Table (PST) maintained by that OE, respectively (this example is developed in detail in Appendix A). The PST in Figure 2(c) contains all the paths admitted by the routing policies from AS1 to the rest of domains in the network according to the AS graph and relationships shown in Figure 2(a). In brief, the PST in Figure 2(c) is the data structure associated with the PSG constructed by an OE in AS1. This same structure shall be used in our prototype implementation.

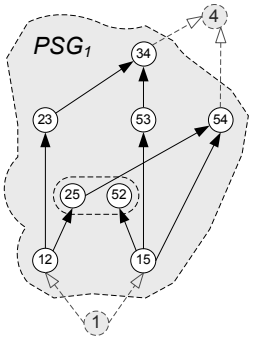
The entries in the PST maintained by each OE are the result of the gathering of information from neighbor OEs through the PSP protocol. The specification of this protocol and its message format is detailed later in Sections 5 and 6.

2.2.2 Next-Hops, Peerings, and IP Prefix Ownership

In principle, the graphs maintained by the OEs do not need to include information about IP prefixes (cf. the table in Figure 2(c)). As mentioned above, the granularity of destinations is kept at the level of autonomous systems, since the goal is to construct graphs of interconnectivity among domains under routing policies. However, the fact



(a)



(b)

	1	2	3	4	5
1	-	(12) [(15),(52)]	[(12),(23)] [(15),(53)]	[(12),(23),(34)] [(15),(54)] [(12),(25),(54)] [(15),(53),(34)]	(15) [(12),(25)]

(c)

Figure 2: (a) An example of an AS graph and the commercial relationships among domains (CP means Customer-Provider whereas PP means Peer-Peer); (b) the PSG constructed by an OE in AS1 (note that all the OEs within AS1 will have the same topological view of the internetwork); (c) the Path-State Table (PST) maintained by an OE in AS1; the PST is the data structure used by the OEs for storing and maintaining the PSGs that they construct—in other words, (b) simply shows the graphical representation of the information contained in table (c). For a more detailed description of this example please refer to Appendix A.

of not binding the address space ownership to ASs may considerably limit the potential use of PSGs for security purposes. For instance, without this binding, it would not be possible to confirm the validity of the following during the overlay inspection:

1. If the first AS in the route was authorized to advertise the IP prefix(es) contained in the updates, and thus can claim ownership of that address space—[origin, prefix(es)] validation.

2. If the router or OE that sent an update was authorized by its AS to advertise the information contained in the update—[next-hop, route] validation, where a "route" pairs destination IP prefixes with paths to those destinations.

3. If the router or OE that sent an update was authorized by its AS to advertise information to the recipient router or OE—[peering] validation.

Although these issues are out of the reach of the prototype that will be implemented and tested in this SRA, we are already preparing the ground to integrate solutions addressing these challenges. Our goal is that the overlay can reject a spectra of unauthorized updates, in a way that can be deployed and used in practice with bearable overhead. The lessons learned from the past are particularly important in this subject, so our approach will be analyzed and discussed in detail with Cisco.

Important remarks regarding the overlay layer — As mentioned above, the OEs can be embedded in the router OS or be built on external (dedicated) hardware, such that each OE can handle one or more BGP processes through xPSP sessions—the processes are distinguished by their IDs. Note that, to avoid potential vulnerabilities, the overlay layer could be set up using a separate address space. Indeed, the overlay could use an address space not routeable in the Internet, since it is desirable that this control layer cannot be reached through the public Internet. In this sense, the overlay can be decoupled from the data plane in a way that BGP does not. Nevertheless, without a dedicated network, the traffic in the overlay will end up sharing resources with the data plane, which, together with the API and the overlay part on the router side, are all issues that require attention as possible targets for attacks.

Another important remark is the issue of transporting control information through the network. As mentioned in [6], we should avoid relying on the routing to convey information about the routing system we want to secure. To solve this issue, our approach is very simple: *there is no routing protocol in the overlay*. In section 4.2, we shall show that the flow of control messages through the overlay follows a chain of concatenated "links", and that our architecture supports that the OEs can dynamically join or leave the overlay without compromising the global distribution of messages through the latter.[a]

[a] Note that in practice the same occurs with BGP, since in general terms, BGP routers can dynamically join or leave the interdomain routing system without compromising the global distribution of messages in the Internet. We will show that for an AS X to join the overlay, an OE $\in X$ needs to set up at least one "link" (an ePSP session) with one OE belonging to each adjacent domain (similarly as current eBGP sessions) and we will describe the process for sending updates throughout the overlay network using these "links". We will also show that, according to compact routing theory, with our update scheme, routing events are transmitted optimally through the overlay on Internet-like graphs.

Moreover, to avoid many potential vulnerabilities in the control information itself, our strategy is also very simple: *the information distributed through the overlay only contains data created, processed, and advertised by the overlay*. In other words, we do not trust BGP for constructing and maintaining the PSGs.

In addition, note that all the OEs inside a domain must be synchronized through the iPSP protocol, and converge to the same PSG, i.e., they must all have the same topological view of the Internet. In the implementation of our prototype, each domain will construct a complete view of the internetwork—though clearly subject to the pruning due to the routing policies. For scalability reasons, it could be the case that a domain X would not want to deal with the data volume and burden of constructing a PSG of the entire Internet. Potential approaches to cope with this issue will be analyzed and discussed with Cisco. Note that this issue might impact on the application of the PSGs for improving the routing security. For instance, if X is a stub domain, then this is not a problem for the Internet routing system; but, if X is a transit domain, it might affect the graphs constructed by X's neighbors, and therefore the validation of routes that go through X. Clearly, if every transit domain runs the overlay protocol without restrictions, then there is no such problem.

3 Inside the Router Operating System

This section specifies the interactions between the overlay and the BGP processes internally in the router OS. We describe the details for Quagga's router implementation [3], but as mentioned above, the specification is sufficiently general to be exported to other router OSs. Quagga is a full fledged routing solution that covers most intradomain and interdomain routing protocols available in the Internet. The Quagga suite is currently in version 0.99.17, which is the one used as a reference in this document. The modular and easily extensible nature of Quagga makes it a suitable candidate for programming and assessing the behavior of the router side of the PSP/BGP protocol suite. To explain the integration with Quagga, we first outline how Quagga processes and manages the `update` of BGP messages. After that, we describe the modifications required to implement the overlay part inside Quagga with minimal effect on the already existing code-base. This is accomplished through a new library in Quagga, the `libPSP`, which hides all the internal details from Quagga's BGP implementation.

3.1 Quagga Internals

Quagga uses an event driven system with internal queues to optimize the BGP processing and to schedule and prioritize the different tasks. Quagga considers a task as any event happening on the router, e.g., the processing of a new update message, a route change, a new BGP peer is connected, and so on. In this section, we focus on the processing of BGP messages, given that this is key not only to reduce the churn and convergence time of the routing system, but also to inspect the updates for security reasons. Figure 3 depicts the current processing in Quagga. First, the BGP message is read (`bgp_read`), and after a sanity check of the content of the update (`bgp_nlri_sanity_check`), the message is parsed (`bgp_nlri_parse`) and different actions are carried out depending whether the packet is an update/withdraw (`bgp_update`) or an explicit withdrawal (`bgp_withdraw`). Note that, in case that the sanity check detects an error, a notification is sent (`bgp_notify_send`) and the BGP session is closed.

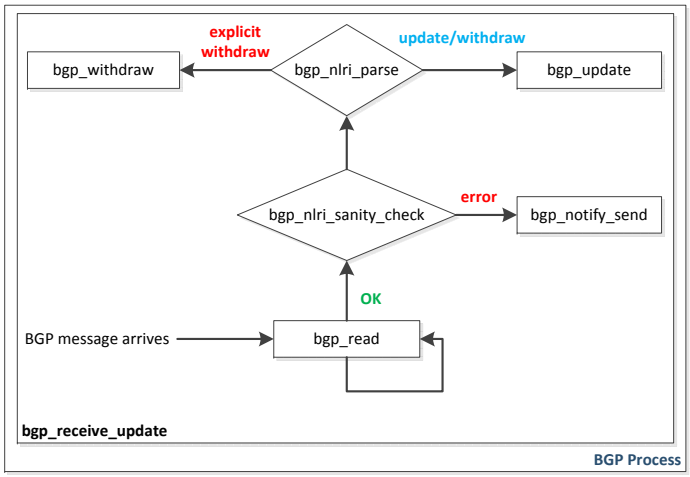


Figure 3: Processing of BGP messages by the `bgp_receive_update` function inside Quagga. The `bgp_update` and `bgp_withdraw` functions are detailed later in Figures 4 and 5, respectively.

Figures 4 and 5 show the processing of an update/withdraw and an explicit withdrawal through the `bgp_update` and the `bgp_withdraw` functions, respectively. In the first case (cf. Figure 4), the function `bgp_update` passes the update through the import filters and performs additional sanity checks in order to prevent issues such as the utilization of invalid prefixes, etc. If the update passes this stage, then Quagga checks if the update corresponds to an implicit withdrawal. If this is the case, the BGP message is scheduled into the tasks queue mainly for aggregation and optimization purposes (e.g., to apply the Minimum Route Advertisement Interval (MRAI) timer). Instead, if the message is a regular update, the internal data structures are prepared through the

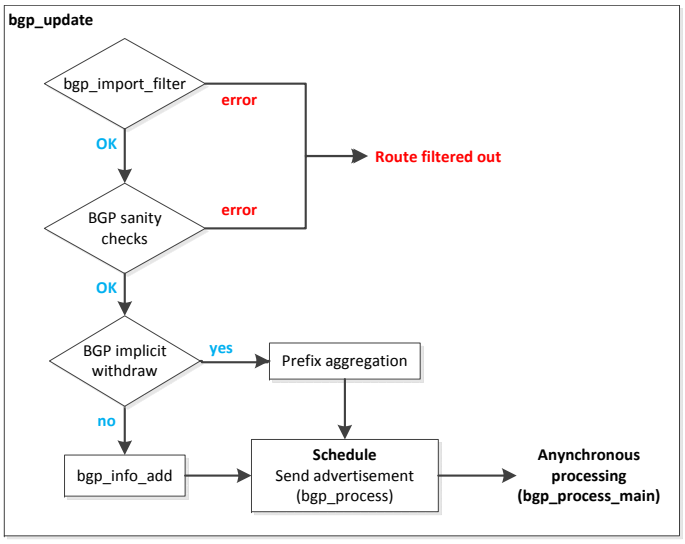


Figure 4: Processing updates and implicit withdrawals: Quagga's `bgp_update` function.

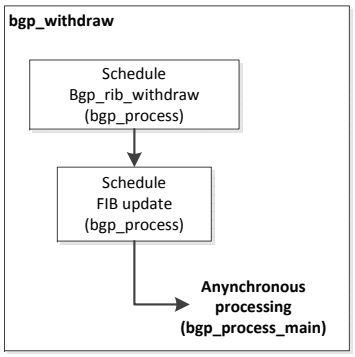


Figure 5: Processing explicit withdrawals: Quagga's `bgp_withdraw` function.

`bgp_info_add` function. Note that the processing flows of implicit withdrawals and regular updates converge in Figure 4, since they are scheduled for running the BGP decision process (`bgp_process_main`) and subsequently update the RIB and/or FIB tables. It is important highlighting that the scheduling process per se is asynchronous, while the process for committing changes to the RIB and FIB is synchronous—though run by a different thread.

As detailed in Figure 5, when the message corresponds to an explicit withdrawal, the router sequentially schedules two events, one for updating the RIB and another for updating the FIB. From Figures 4 and 5 it can be noted that, both updates/withdraws and explicit withdrawals are finally scheduled in the same queue, and attended by the same callback function, the `bgp_process_main`. The latter is shown in Figure 6. This function runs the BGP decision process (`bgp_best_selection`) and decides whether the `update` message changes the best route. If the current route remains unmodified, then no further actions are required ("Old best" → Exit). However, if the outcome of the BGP decision process changes the best route, then the routes that must be updated and/or removed from the RIB are first marked, and then changed and advertised to the corresponding BGP peers according to the export policies configured on the router. The `bgp_process_main` function also commits the changes to the FIB.

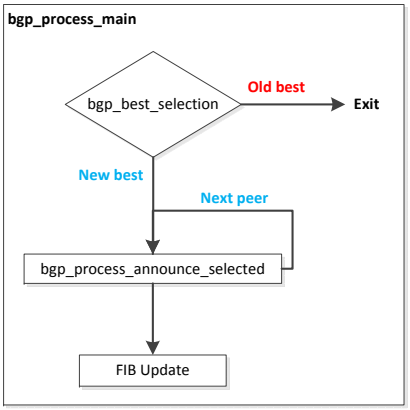


Figure 6: Main callback for modifying the FIB/RIB after processing either an update/withdraw or an explicit withdrawal.

We now proceed to describe the modifications proposed in Quagga in order to integrate the overlay inspection and control of BGP messages with minimum effort.

3.2 Overlay Operations in Quagga (the libPSP library)

Figure 7 shows a more detailed picture of the elements introduced in Figure 1 as well as their corresponding interactions. The figure shows the router, the API, and the overlay layer. The latter has two components, a new library on the router OS side called `libPSP`, and the Overlay Entity (OE).

Note that the current `bgp_receive_update` function in Quagga (see Figure 3) is now split into two sub-functions, namely, the `bgp_receive_update` and the `bgp_receive_update_new` functions in Figure 7, which combined, perform exactly the same tasks as formerly described in Figure 3. The advantage of this approach is that the functions shown in Figures 4, 5, and 6 remain unmodified, so the only change needed in the BGP code is the API to an intermediate process that lies between the `bgp_nlri_sanity_check` and the `bgp_nlri_parse` functions in Figure 3. This intermediate processing is performed by an overlay control layer that acts transparently to the BGP functions—by transparency we mean that the communication between the `bgp_nlri_sanity_check` and the `bgp_nlri_parse` functions as well as their internal processing at the BGP level also remain unmodified. In other words, except for the introduction of the new API, the functionality of BGP is neither changed by nor aware of the intermediate inspection and processing performed by the overlay layer.

It is worth noting that in case that the overlay layer is either shutdown or disabled, the `libPSP` provides a "short circuit" between the `bgp_nlri_sanity_check` and the `bgp_nlri_parse` functions, guaranteeing in this way the continuity and the usual processing flow of BGP messages inside the router (see the flow through the `OV_DISABLED` arrow in Figure 7).

The new library is a lightweight process in the router OS that enables the inspection and potential processing of update messages by the OE transparently to BGP. More specifically, after the `bgp_nlri_sanity_check` of a BGP update, the `ov_receive_update` function of the `libPSP` library is invoked through the API. The task of this function is basically to enqueue the update so that it can be transmitted to the OE through the xPSP protocol, and then exits. After the inspection of the update, the OE responds to the `ov_process_update` function (see Figure 7), and can return four different values: `OV_DISCARD`, `OV_OK_NORMAL`, `OV_OK_UPDATE_TABLES`, or `OV_OK_NO_ACTION`.[1] An `OV_DISCARD` value is returned when the update is not legitimate and thus must be discarded with no further actions required from the router side (e.g., the update is rejected during the path or the origin inspection). An `OV_OK_NORMAL` value is returned when the update is legitimate and has to be handled by BGP following the usual processing flow. Examples of this latter case are any legitimate BGP update that does not imply changes in the PSG, such as updates due to Traffic Engineering actions, changes on policies, the failure of one link between two multi-connected domains, etc. Note that the return of an `OV_DISCARD` or an `OV_OK_NORMAL` value indicates that the role of the overlay for that particular BGP processing thread was only to perform inspection, and the validation or refusal of the update.

[1] The functions `ov_receive_update` and `ov_process_update` in Figure 7 are described in more detail later in Figure 9 and particularly in Section 3.3.

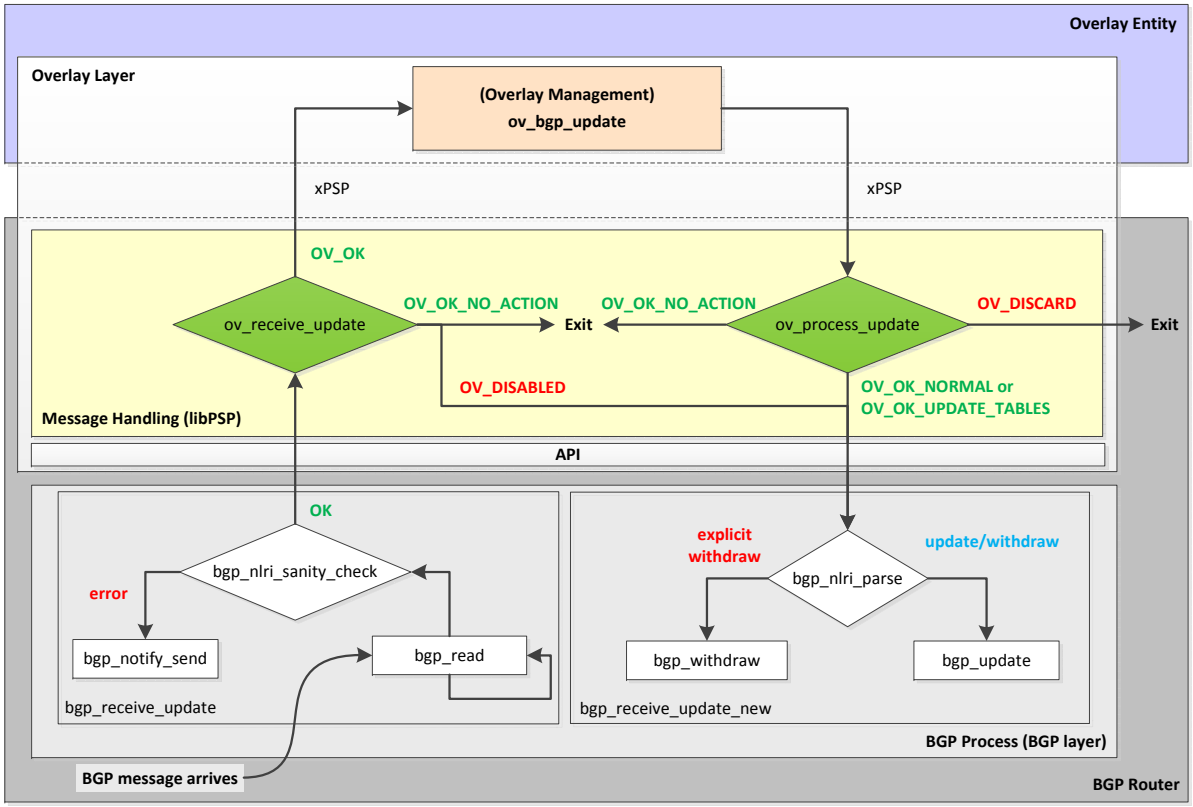


Figure 7: Overlay operation within Quagga and interactions with the Overlay Entity (OE). Note that the processing of BGP messages inside Quagga (see the `bgp_receive_update` function in Figure 3) is now split in two sub-functions, namely, the `bgp_receive_update` and the `bgp_receive_update_new` functions, which interact directly with the message inspection and handling provided by the `libPSP` library.

The `OV_OK_UPDATE_TABLES`, on the other hand, is returned when the update is legitimate, but it was the overlay the one in charge of its handling. For instance, consider the case when a link directly connected to a router, R_1, fails affecting the PSG. Figure 8 shows the details of how the `libPSP` detects the link failure directly from the kernel. Observe that the `Zebra status notifier` alerts the `ov_interface_monitor` function in the `libPSP` library about the change in the physical interface. Each interface has associated an `if_status` flag, which is used to trigger a message to the overlay entity configured in router R_1—let us assume OE_1 in this case. In practice, the detection of the failure at the kernel level occurs long before that the first BGP message containing the routes that need to be withdrawn arrives to its corresponding queue. Since the overlay layer is notified of the link failure, it can correlate and even filter all the BGP updates locally triggered by this event. This filtering process eliminates the unnecessary path explorations that would be performed by other routers in the network, e.g., when one or more destinations become unreachable through AS-paths that are all affected by the local failure. More specifically, if the link failure affects the PSG computed by OE_1, then the overlay layer takes complete control of the event, and performs a series of tasks, including the following:

- OE_1 recomputes the path-state graph PSG_1 and advertises the topological changes using the overlay protocol (iPSP/ePSP);
- OE_1 mandates the removal of the routes affected by the failure from the local BGP RIBs and FIB;
- the `libPSP` intercepts and filters all subsequent messages locally generated by BGP due to the failure. The goal is that the local overlay processes (i.e., the `libPSP` and OE_1) shield the routing system from the flood of BGP messages that would be generated by the link failure.

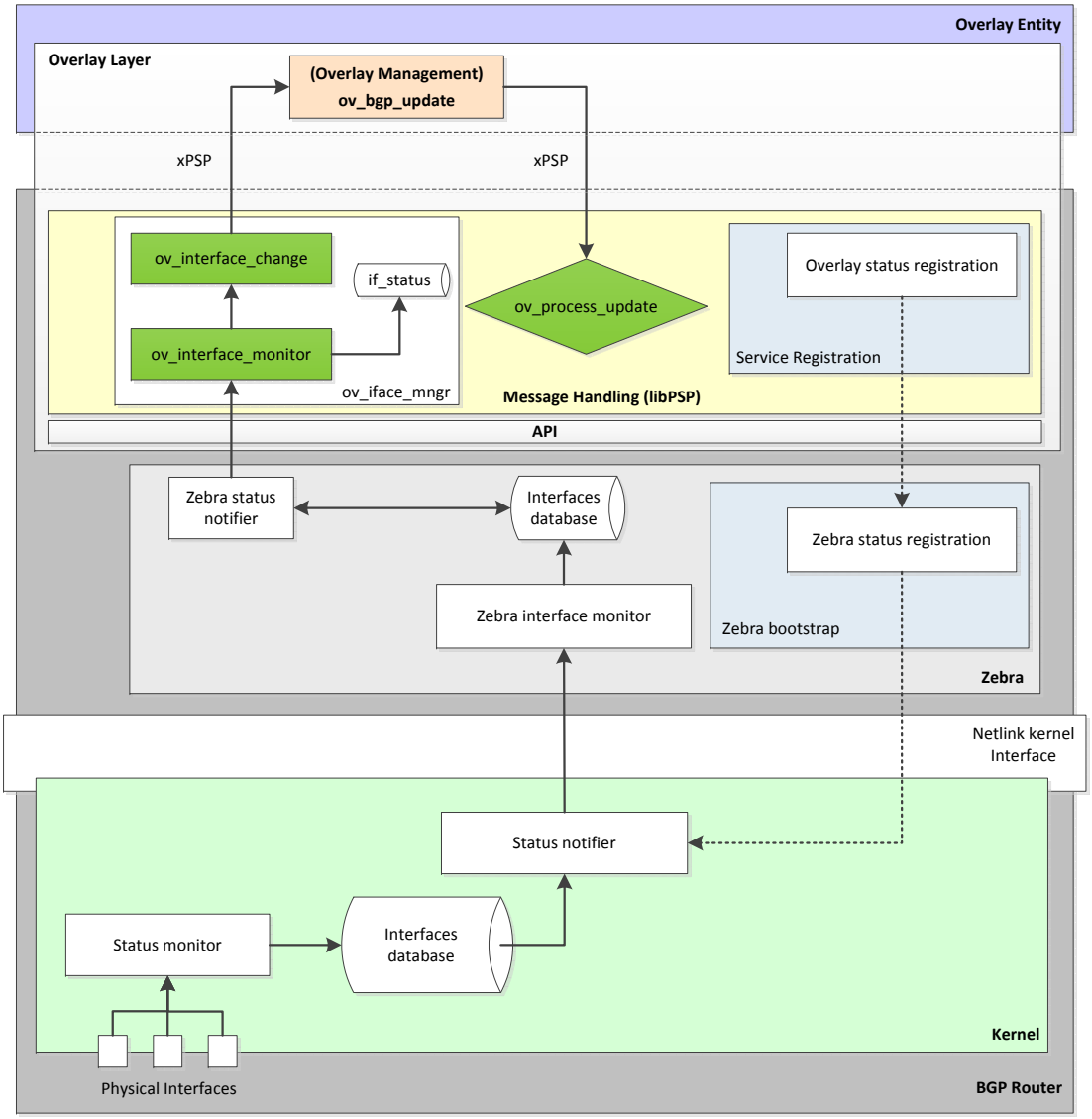


Figure 8: Detecting the status of the links directly from the kernel. The registration and bootstrap processes are described later in Sections 3.4, and 4.3, respectively.

It is important to notice that OE_1 will recompute its PSG, and advertise the topological changes according to the usual export policies of routing domains. This means that the list of OE neighbors contacted by OE_1 will depend whether the link that failed is a peer-peer or a customer-provider link. All the OEs contacted will recompute their PSGs and advertise the new PSGs accordingly. After recomputing the graph, each OE can individually infer which AS-paths are affected by the failure, and thus instruct the removal of the corresponding routes from the BGP RIBs and FIBs. A simple way to do this is could be to use AS-path filters—though this particular subject and its potential optimization will be discussed in detail with Cisco. An initial approach could be the following. An OE receives an update through the ePSP protocol containing topological information that implies a change on its PSG (e.g., a link between two ASs has been removed). The OE recomputes the PSG, prepares an AS-path filter, and schedules the update so that BGP commits the changes. After the filter is enforced by BGP, the routes affected by the change are removed from the routing and forwarding tables. If BGP messages are subsequently generated by these changes in the RIB and FIB, they will be intercepted and discarded by the `libPSP`, since in this case, the convergence at the AS level will be handled entirely by the overlay layer. The OE will then advertise the changes to the corresponding OE neighbors, and finally instruct BGP to remove the filter, leaving the router in the same conditions as before the topological update was received.

In previous reports, we have shown that the processing and control provided by the overlay layer can make the routing system converge several orders of magnitude faster than with BGP. In Section 4.2, we will show that this can be accomplished with $O(n)$ ePSP update messages (being n the number of ASs in the network), which, according to compact routing theory, it is optimum on Internet-like topologies. *Indeed, the update and convergence of the routing system in cases that entail changes in the PSGs can be handled entirely by the overlay layer, i.e., without triggering a single BGP message from the routers*.

To explain in more detail how can this be made with only minor modifications and using the API on the router side, consider the interactions shown in Figures 8 and 9 jointly. Upon the detection of a local link failure (cf. Figure 8 and step 1b in Figure 9), the OE determines whether the PSG is affected, and if so, it sends a message to the `ov_process_update` function in the router through the xPSP protocol (step 2 in Figure 9) instructing the updates required in the local RIB and FIB. The `ov_process_update` schedules the updates for the RIB and FIB and returns an `OV_OK_UPDATE_TABLES` value (see steps 1b $\rightarrow$ 2 $\rightarrow$ 3a $\rightarrow$ 4a $\rightarrow$ 5b in Figure 9). BGP will then commit the changes in the RIB and FIB and possibly schedule routing updates that need to be sent to certain BGP neighbors. In that case, all those updates will be intercepted, correlated, and discarded by the overlay layer, which will return an `OV_OK_NO_ACTION` value indicating that no further actions are required from BGP (see the `OV_OK_NO_ACTION` value returned by the `ov_sanity_checks` function in Figure 9). In brief, the overlay layer at the origin router (i.e., the router directly connected to the link that failed) will make BGP "believe" that its messages are sent to BGP neighbors as usual.

It is important to observe that, if a link failure does not affect the PSG computed by the OE, then the OE will only inspect the BGP messages triggered by the failure and return an `OV_OK_NORMAL` value, giving back the control to BGP for the usual processing and advertisement of BGP messages (steps 1b $\rightarrow$ 2 $\rightarrow$ 3a $\rightarrow$ 4b in Figure 9). In that case, all the OEs along the flow of advertisements will act accordingly, i.e., perform the inspection and return an `OV_OK_NORMAL` value to BGP.

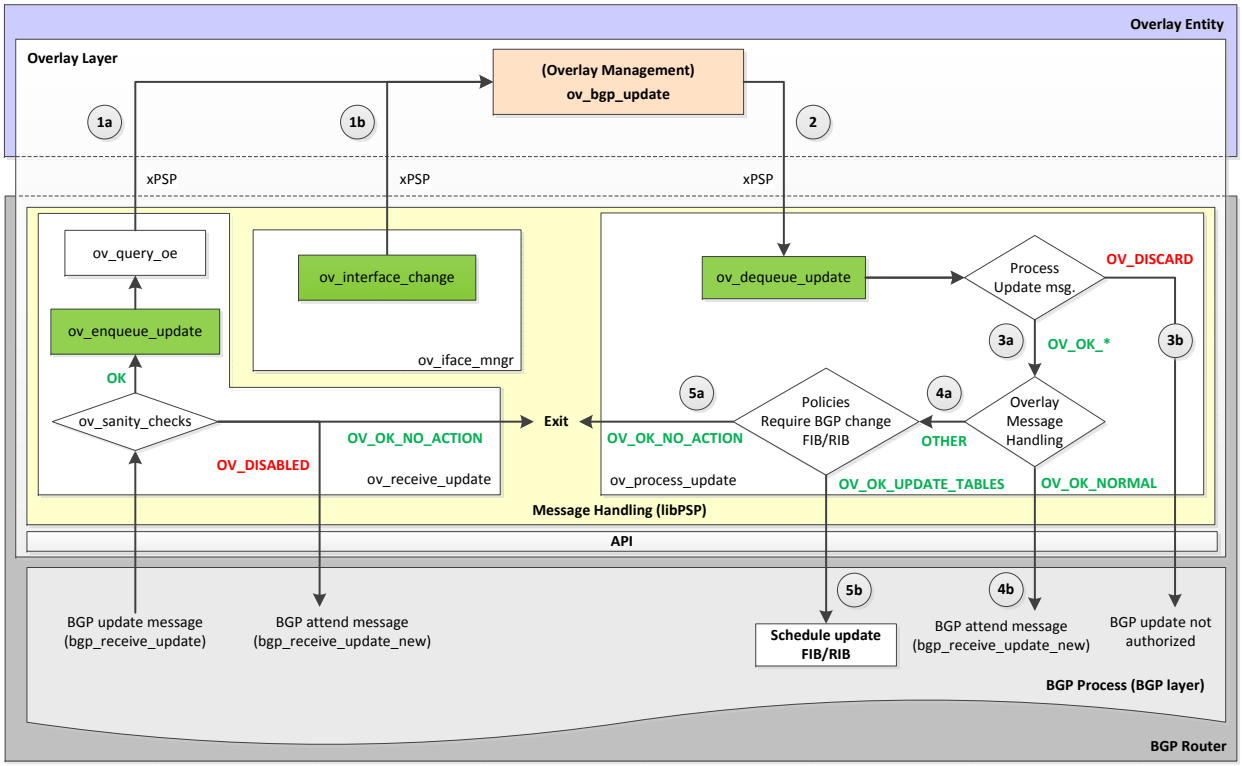


Figure 9: A more detailed description of the interactions and processing flows between Quagga and the OE. Observe that now we show the internal structures of the `ov_receive_update` and `ov_process_update` functions in Figure 7. Also note that the processing through 1a is triggered upon receiving a BGP update message, whereas the one through 1b is triggered upon changes on a router interface.

Another very important observation is that BGP operations are never stalled by the overlay layer, since the interactions between BGP and the overlay are restricted to calls and message queuing.

Figure 10 summarizes the global Finite State Machine (FSM) for processing BGP messages. This FSM captures the interactions between the BGP and overlay processes inside Quagga as well as the interactions with the OE.

3.3 API with Quagga

The interactions between the overlay and the router are carried out through a set of public functions that make available the operations permitted between the router OS and its embedded overlay interface. For the sake of simplicity, we keep this public interface to a reduced set of all the helper functions that will be part of the library. Such public interface is composed of:

- **int** `ov_register`(**struct** `sockaddr *overlay_entity`)
  - **Purpose:** Registers the BGP process to the Overlay Entity
  - **Return value:** 0: Success, 1: Error.

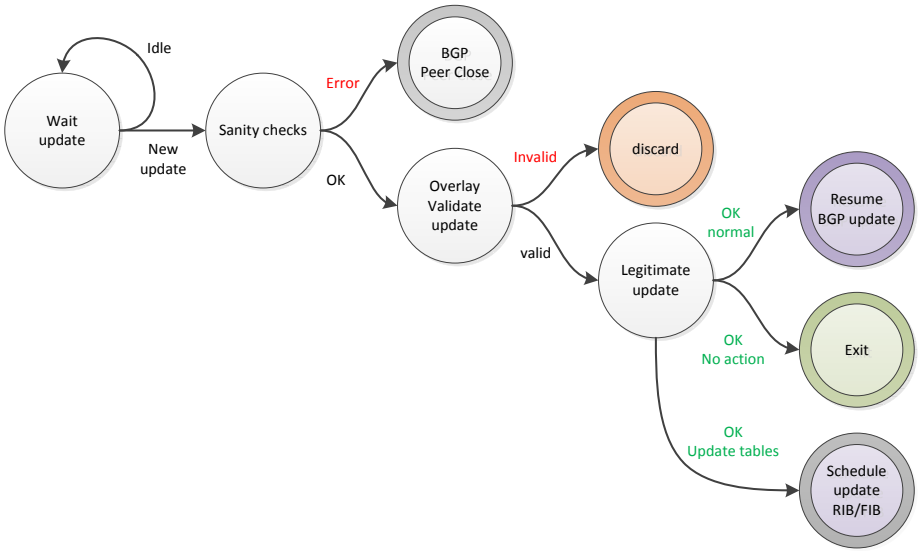


Figure 10: Finite State Machine (FSM) summarizing the processing of update messages and the interactions between BGP and the overlay layer.

- **int** `ov_unregister`(**struct** `sockaddr *overlay_entity`)
  - **Purpose:** Closes the connection with the overlay and disables the PSP functionality within a BGP routing process.
  - **Return value:** 0: Success, 1: Error.
- **int** `ov_receive_update`(**void** `*bgp_message`)
  - **Purpose:** Forwards to the overlay entity the BGP update received. It can also selectively intercept and discard BGP updates if they are correlated to a recent link failure.
  - **Return Value:** `bgp_message`: to the OE, `OV_OK_NO_ACTION`: OK (but no further actions are required from BGP since the overlay has already taken control of the processing), `OV_DISABLED`: BGP should continue its normal operations since no OE is registered.
- **int** `ov_process_update`(**void** `*message`)
  - **Purpose:** It receives messages from the overlay entity and decides the actions to be taken within the router.
  - **Return Value:** `OV_DISCARD`: Discard (invalid update), `OV_OK_NORMAL`: OK (BGP continues processing the update as usual), `OV_OK_NO_ACTION`: OK (but no further actions are required from BGP since the overlay has taken complete control of the processing), `OV_OK_UPDATE_TABLES`: OK (schedules the changes required in the RIB/FIB).

- **void** `ov_iface_mngr`(**void** $*$ `zebra_link_info`)
  - **Purpose:** Intercepts changes in the status of the links within the BGP router. This allows to optimize the case when a link fails, speeding up the process of requesting the OE the re-computation of the PSG and the removal of the affected routes from the RIB and FIB tables.
  - **Return Value:** No return value is specified in this function.
- **struct** `ov_status` $*$ `ov_get_status`()
  - **Purpose:** Queries the status of the overlay in terms of internal statistics.
  - **Return Value:** Descriptor of the status.

Note that the `ov_receive_update` and `ov_process_update` functions together enable the inspection and the potential validation of the legitimacy of the BGP messages received. To this end, the OEs could reasonably perform the following actions:

- Inspect if the BGP router or the OE that sent the update on behalf of the neighbor domain is authorized to perform this action.
- Inspect if the routes contained in the update are feasible by means of the PSG.
- In case of a topological change, inspect the legitimacy of the change sent through the overlay layer.
- Inspect if the AS originating the message has the right to advertise or withdraw the IP prefix(es) contained in the updates.

The architecture and interactions described so far between BGP and the overlay layer are not sufficient to legitimate the outcome of the overlay inspections, but clearly provide a reasonable basis on which to build. It is important to recall the impact and overheads introduced by former BGP security solutions. In particular, Zhao et al. [7] showed that all the variants of S-BGP significantly increased the convergence time. We have already shown in previous reports that, by taking advantage of our PSGs, we can reduce the convergence time and churn of routing updates by several orders of magnitude in the Internet. This suggests that it might be possible to integrate the overlay inspections into the routing system and still maintain acceptable convergence times.

3.4 Overlay Registering and Management Console

To initialize the overlay process as well as to have access to the functionalities offered by the overlay, it is necessary to register the user interface into Quagga. To this end, the management console provides a set of new commands that permit to register, interact, and disable the PSP functionality and its support.

The supported set of commands is the following:

- **psp register** $<$ *overlay_entity* $>$: used in order to configure the OE responsible for a BGP process.
- **psp status**: outputs the status of the overlay entity.
- **psp close**: used to disable the activities provided by the overlay.

4 The Overlay Entities (OEs)

We provide now a detailed description of the interactions among the Overlay Entities (OEs), including a proposal for creating the overlay layer and the type of control information exchanged between the OEs. We anticipate that the overlay layer does not carry routing information per se (such as the best route toward a destination), but instead carries control information aimed at inspecting BGP updates and improving different aspects of the BGP protocol. In this SRA, we focus on the testing of the convergence and churn properties of the PSP/BGP protocol suite, so in our initial prototype, the BGP updates will always pass the inspections performed by the OEs.

4.1 The Overlay Layer: Construction and Updates

We first outline the intuition and requirements for constructing the overlay layer and keeping the information updated, and then introduce the notation used and formalize the overall construction and update processes in Section 4.2.

It is important highlighting that the construction and management of the overlay proposed in this document targets to:

1. optimize the distribution of control messages through the overlay;
2. and provide the means to distribute these messages without the need of a routing protocol in the overlay (this is to avoid the issue of relying on routing to secure a routing protocol [6]).

With this approach, it is possible to reach the lower bound $\Omega(n)$ in the number of control messages for a network of size n (see compact routing theory on Internet-like topologies [8]), without actually needing a routing protocol in the overlay.
In summary, the construction and management processes proposed in this SRA are focused on the distribution of control messages. Note, however, that the functionality and the end purpose of the overlay should not be constrained by these decisions, so other approaches may be considered both for constructing the overlay and for maintaining the information updated.

To illustrate the construction process, consider the example shown in Figure 11(a). The figure shows a graph of interconnectivity among ASs, where the dark nodes represent Tier-1 domains, while the nodes in light gray are either transit or stub domains. For simplicity in the exposition, let us assume that each AS in Figure 11(a) contains a single OE.[2] In this framework, an overlay entity OE_x in a domain X will establish ePSP sessions with OEs in each neighbor AS, and only with its neighbors (i.e., only with OEs belonging to domains that are directly connected to domain X). These sessions could represent either *primary* or *secondary* connections to the overlay layer—the difference between these two types of ePSP sessions and their roles will be described later in this section.

By construction, the Tier-1s are interconnected in a full-mesh in the overlay layer, so each Tier-1 maintains active a connection with the other Tier-1s in the network. All the connections between Tier-1s are primary connections. The rest of domains choose

[2] In practice there will be one OE per BGP router inside an AS, which will be interconnected through the iPSP protocol. Notwithstanding, all the OEs within a domain must have the same topological view of the network (i.e., they must "see" the same PSG), so, without loss of generality, we can use the single-node abstraction in order to simplify the explanations that will follow.

one, and only one among their providers, as their primary connection in the overlay. This means that each domain X, such that X is not a Tier-1, will have only one primary connection to the overlay, and $(\mathcal{P}(X)-1)$ secondary connections, where $\mathcal{P}(X)$ denotes

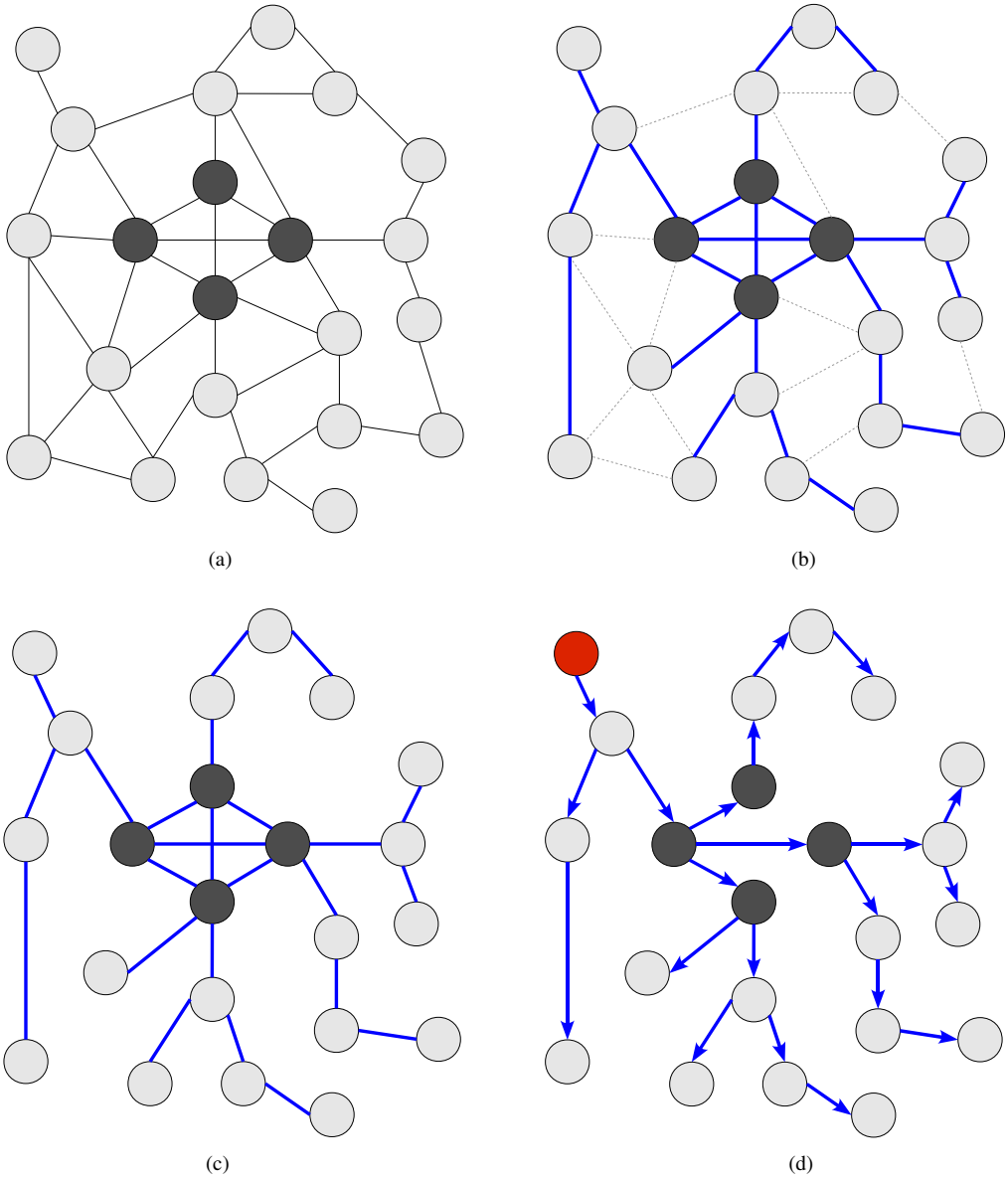


Figure 11: (a) Graph of interconnectivity among ASs, where the dark nodes represent Tier-1 domains, while the nodes in light gray are either transit or stub domains; (b) primary sessions at the overlay level between neighbor domains; (c) subgraph with the primary sessions, which is composed of disjoint trees rooted in the Tier-1 domains; (d) flow of control messages through the overlay network for an event that needs to be advertised to all domains (in the example the event triggering the updates is originated by the domain located at the top-left). Later in this section, we shall show that by following a set of simple rules, the flow of messages can be forwarded without the need of a routing protocol, and that the number of messages required is $O(n)$, which is the lower bound that can be reached with compact routing for an Internet of size n [8] .

the number of providers of domain X (see Figure 11(b)). By following this strategy, the primary sessions in the overlay can be abstracted as a set of trees, each of which has its root placed in a Tier-1 domain (see Figure 11(c)). More specifically, each Tier-1 represents the root of a tree that links a set of neighbor (customer-provider) OEs, where the hierarchical structure of customer-provider's relationships in the Internet guarantees the acyclic nature of the primary trees. Note that, by construction, trees rooted in different Tier-1 domains are node disjoint. Also note that the average AS-path length in the Internet has remained almost constant at ≈ 4.2 over more than a decade [9], so if domains tend to establish their primary sessions with their largest providers, the average length of the primary tree branches will be very short. In brief, the overlay layer is composed of OEs that are interconnected following the physical topology, such that the set of primary ePSP sessions actually represents a subgraph of the AS graph (cf. Figure 11(c)).

Before proceeding to describe the roles of the primary and secondary sessions, it is worth making the following remarks. When a new domain X joins the overlay, an OE in X establishes a set of " links" (ePSP sessions) with OEs in each of X's neighbor domains according to the following criteria:

- The sessions with OEs in X's providers are all secondary except for one, which is chosen by X as its primary session.
- The sessions with OEs in X's peers are all secondary, except between Tier-1 providers, which are always primary sessions.
- The sessions with OEs in X's customers may be either primary or secondary depending on the preference of each customer.
- The control information initially exchanged through "*each*" of these sessions is composed of:
  - The PSG graphs filtered according to the routing policies used in the Internet (see Appendix A). This information can be used both to reduce the BGP convergence time and churn of updates as well as for AS-path inspection and authentication purposes.
  - The IP mappings to bind the IP address space ownership to the corresponding AS. This information can be used for origin inspection and authentication.
  - The set of authorized next-hops in order to inspect if the router or the OE that sent an update was authorized by its AS to advertise the information contained in the update.
- After the initial exchange of information, only incremental updates are sent through these sessions; and, as we will describe below, depending on the event, the routing policies will determine whether the updates should be forwarded through the primary sessions or the secondary ones.

Clearly, the primary and secondary sessions are configured in the OEs. Recall that, for simplicity in the exposition, we assumed a single node abstraction (one domain $\leftrightarrow$ one OE), but in practice these ePSP sessions will be distributed among a set of OEs within a domain, which will synchronize their control information and will converge to a consistent topological view of the internetwork through internal iPSP sessions.

Primary ePSP Sessions — As mentioned above, when a domain X joins the overlay there is an initial exchange of information between X and its neighbors (both through the primary and secondary ePSP sessions), and after that only incremental updates are exchanged between neighbors. In particular, events that need to be communicated to "all" domains in the overlay are always disseminated through the primary sessions. An example of this is shown in Figure 11(d). This approach not only enables the distribution of information optimally in the overlay [8], but most importantly, it can be implemented without actually needing a routing protocol. A way to achieve this is the following:

1. When a domain X is the origin of an event that needs to be advertised to the entire overlay, X sends updates through all its primary ePSP sessions (e.g., X has now a new customer, so a customer-provider link needs to be added to the PSGs). Observe that by "all its primary ePSP sessions" we mean: all primary customers and one primary provider if X is not a Tier-1, and all primary customers and the other Tier-1s in case X is one of the latter.

2. When X's neighbors receive the update, they process the information (e.g., they recompute and update their PSGs), and prepare new messages to be sent through their own primary sessions, excluding the one with domain X and the ones with other Tier-1 when the update is sent between two Tier-1s (see Figure 11(d)).

3. The process is repeated until all domains in the network are updated.

As it can be noted, the forwarding process is straightforward, since each OE only needs to send the control messages to a set of predefined primary neighbors according to the above rules. Clearly, this can be implemented without the need of a routing protocol in the overlay. Moreover, it can be easily shown that the number of control messages needed to update an overlay of size n through the primary sessions is $O(n)$.

Secondary ePSP Sessions —The secondary sessions, on the other hand, are needed: i) for exchanging information exclusively involving the two neighbor domains (e.g., to exchange the set of authorized BGP next-hops between two neighbor domains); ii) for backup purposes (e.g., when a primary ePSP session fails[3]); iii) for disseminating information upon changes involving peer-peer BGP sessions (e.g., a new peer-peer relationship is established between two ASs, the physical connection between two peers fails, etc.). In this latter case, the secondary sessions are used for disseminating events that need to be advertised only to a subset of OEs in the network. For instance, consider the example shown in Figure 12(a), where the peer-peer connection between AS1 and AS2 fails. When this occurs, both AS1 and AS2 will update their corresponding PSGs, and after that, the topological change must be advertised downstream, i.e., only to AS1's and AS2's customers (but not upstream to their providers or their peers). Figure 12(b) shows the set of primary and secondary sessions of downstream domains. From Figures 12(b) and 12(c) it can be observed that, the primary sessions with customer domains cannot guarantee that all customers downstream can be notified of the change in the PSG. This is because some domains such as AS4 and AS6 may have primary sessions with domains which are not even aware of the existence of the link between AS1 and AS2 (such as is the case of AS9 and AS10). In the example, AS4 and AS6 cannot be reached from AS1 and AS2 through downstream primary sessions—though they will be clearly reachable through upstream primary sessions.

[3]We will describe this in detail later in Figures 13(c) and 13(d).

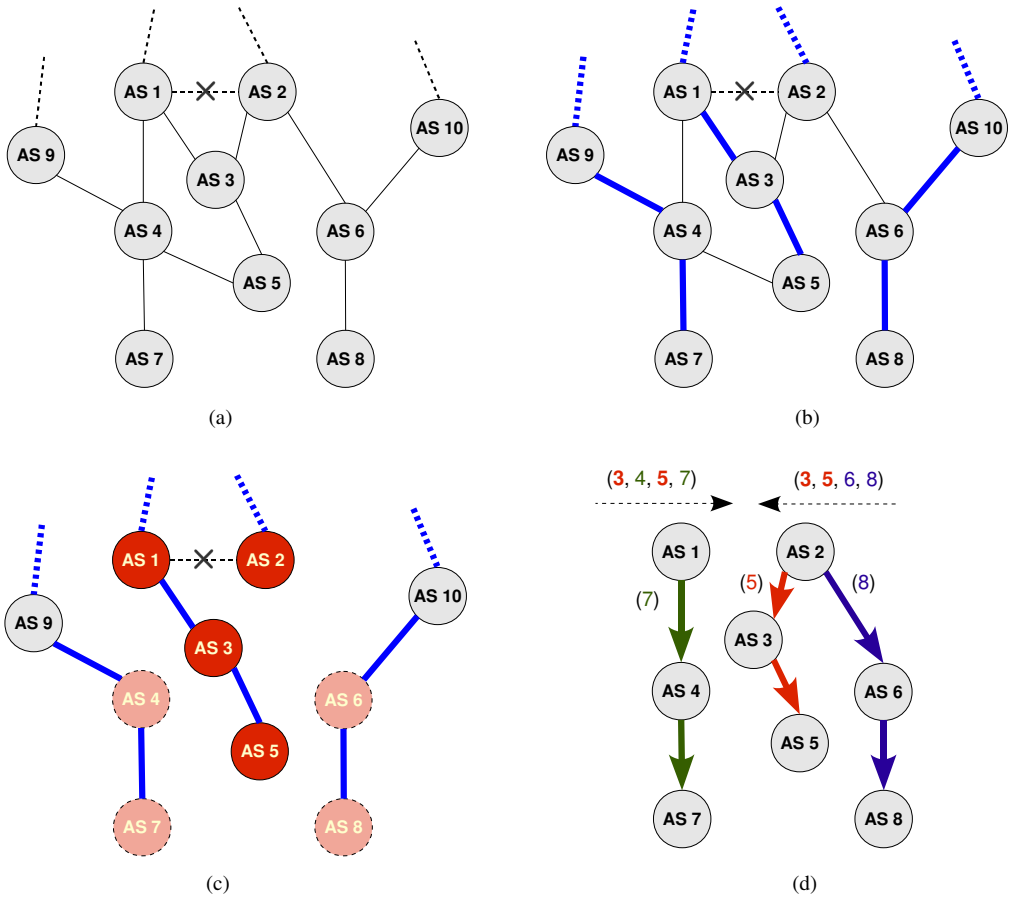


Figure 12: (a) The connection between two peer domains AS1 and AS2 fails, affecting the PSGs constructed by their customers—note that the PSGs constructed by the providers or other peers both of AS1 and AS2 will not be affected by the failure; (b) the set of primary ePSP sessions downstream when the failure occurs; (c) the primary sessions with customer domains cannot guarantee that all customers downstream can be notified of the change in the PSG; (d) a simple solution to handle updates involving peer-peer relationships is that AS1 and AS2 use both the primary and secondary sessions with their direct customers to advertise the change. In the example, each peer will notify its single customers, and since AS2 has the token, it is the one in charge of sending the updates to the customers in common, AS3 and AS5.

Several strategies can be devised to handle the peer-peer relationships in a straightforward way. One alternative is to simply send updates through all the primary and secondary customer sessions, and repeat this process downstream until all customers are updated. Note that the forwarding of updates can be made without the need of a routing protocol in the overlay, though at the cost of increasing the number of control messages due to the flooding process. Another simple solution is shown in Figure 12(d). Both AS1 and AS2 know which domains are advertised solely by one of them through the peer-peer link as well as the ones that they have in common. In the example (see Figure 12(a)), AS4 and AS7 belong only to the PSG advertised from AS1 to AS2, whereas AS6 and AS8 are only present in the PSG advertised from AS2 to AS1. On the other hand, AS3 and AS5 are present in both PSGs. If the overlay protocol establishes that each domain will notify its single customers, and that a token is passed between the

peers for the ones in common, then: AS1 will notify AS4 and AS7; AS2 will do the same for AS6 and AS8 plus the domains that they have in common (AS3 and AS5) assuming that it is AS2 the one that has the token. To this end, AS1 and AS2 could send updates downstream containing the list of domains that must be notified next (see Figure 12(d)). These lists can be precomputed based on the PSGs individually constructed by AS1 and AS2. Overall, by using both the primary and secondary sessions with their direct customers, AS1 and AS2 could advertise the change downstream without the need of a routing protocol. Also note that compared to the former strategy, the latter may considerably enhance the efficiency in the distribution of control messages.

In any case, the strategy for advertising changes involving peer-peer relationships will be discussed in detail with Cisco.

Basic Requirements and Remarks:

- As is usually the case with eBGP, a direct "link" between two peering OEs is required for the ePSP sessions, if not, routing will be needed to forward the overlay messages between adjacent OEs. An initial possibility for testing could be to use secure tunnels for establishing the ePSP sessions.
- It is worth highlighting that the OEs do not need to maintain a graph of the overlay per se; they only need to maintain active a set of primary and secondary sessions with OEs located in their neighbors. In other words, we do not maintain state of the overlay network. Each OE is only aware of its neighbors and the control information conveyed through the overlay.
- The OEs must be able to securely exchange information about: i) the AS-level valley-free paths in the internetwork; ii) the mappings binding IP prefix ownership to AS numbers; and iii) the next-hop information both for adjacent BGP routers and OEs.

4.2 Managing the Control Information Through the Overlay

Before we go into the details, recall that the physical links and the router-level of details are omitted in the PSGs, as we only consider the presence or absence of edges among ASs based on the forwarding policies. In our prototype, the transmission of update messages through the overlay occurs upon changes in any of the following cases: a) when the PSG information must be updated; b) when the ownership of IP prefixes must be updated; or c) when the list of authorized next-hops (either a BGP router or an OE) must be updated.[a] Any BGP event which does not entail one of these changes will be simply inspected and processed locally without triggering updates in the overlay. For the moment, we maintain the complexity of our implementation at this level, but it is worth emphasizing that the dissemination of more sophisticated messages through the overlay can be devised, such as the possibility of using blacklisting against malicious behaviors, and more.

[a] Along this section we will discuss that while some of these changes must be notified to the entire overlay, others only need to be advertised between neighbor domains. In particular, we will discuss a practical approach for handling IP prefix ownership. At this stage, we aim at improving BGP security in a way that can be deployed in practice, rather than considering highly complex infrastructures targeting almost perfect security with questionable overheads.

A simplified version of the steps followed by an AS to join the overlay as well as the control information exchanged during this process are detailed in Algorithm 1 (the notation used is defined in Table 1). First, the primary provider must be chosen, and an administrative process starts for acquiring a digital certificate and a public/private key pair to validate all the operations performed through the overlay (see step 5 in Alg. 1). Although the key distribution system is beyond the scope of this SRA, an option could be to use an approach similar to the one described by White in [6], where the keys and certificates are distributed through a web of trust by a set of Certificate Entities (EntityCerts). These EntityCerts could be implemented by means of specialized overlay entities, which could delegate (e.g., in a hierarchical manner) the creation of certificates and keys as well as their distribution. Indeed, one possibility to support this delega-

Algorithm 1 AS X joins the overlay

Input: $\{\mathcal{P}(X), \mathcal{C}(X), Pe(X)\}$

if $\mathcal{P}(X) \neq \varnothing$ **then** $\{X$ is not a Tier-1$\}$
  $P_X^* \leftarrow$ selectPrimaryProvider($\mathcal{P}(X)$)
end if
5: $getCertificate$
$CreateOverlayEntity(OE_X)$
for all neighbor $Y \leftarrow \{\mathcal{P}(X) \cup \mathcal{C}(X) \cup Pe(X)\}$ **do**
  if $Y = P_X^*$ || $X =$ selectPrimaryProvider($\mathcal{P}(Y)$) **then**
    $CreatePrimary\_ePSP(Y)$
10: $\mathcal{C}_X^* \leftarrow \mathcal{C}_X^* \cup Y - \{P_X^*\}$
  else
    $CreateSecondary\_ePSP(Y)$
  end if
  sendPSG(Y)
15: getPSG(Y)
  sendPrefixMapping(Y)
  getPrefixMapping(Y)
  sendNextHops(Y)
  getNextHops(Y)
20: getACK(Y)
end for

Symbol	*Description*
X	A domain in the network
$\mathcal{P}(X)$	Set of providers of domain X
$Pe(X)$	Set of peers of domain X
$\mathcal{C}(X)$	Set of customers of domain X
OE_X	An Overlay Entity of domain X
P_X^*	Primary provider of domain X
$\mathcal{C}_X^*$	Set of customers which have domain X as their primary provider

Table 1: Notation.

tion could be that the Tier-1s hold the root certificates, from where the web of trust could be constructed. Observe that, in this way, the authorizations to issue certificates could follow the primary branches through the overlay, so a distinctive factor while selecting the primary provider could be the capability of this latter to issue and delegate certificates. Even though this strategy can reduce the overhead and facilitate the administrative tasks to get the certificates and keys, the decision of whether to exploit the overlay for this purpose or use an external solution shall be discussed in detail with Cisco.

Once the certificate has been obtained, an overlay entity must be created (OE_X), which can now start establishing ePSP sessions with its neighbors. Through each of these sessions, the OEs initially exchange information about their topological views (i.e., their PSGs), the pool of IP prefixes originated by their corresponding domains, and the set of authorized next-hops (see steps 14 to 19 in Algorithm. 1). After the initial exchange of control information, only incremental updates are sent through the overlay. In our initial prototype, the functions sendPrefixMapping(Y) and getPrefixMapping(Y) in steps 16 and 17 of Algorithm. 1, allow that two neighboring OEs exchange the IP prefixes owned by their corresponding domains, providing in this way a mapping between a neighbor AS number and the pool of IP prefixes originated by that neighbor. For the moment, we are maintaining these mappings only between neighbor domains. Such a scheme could offer a reasonable trade-off between the level of security obtained and the complexity required for route origin attestations, since the inspection and validation of the origin in BGP updates could be based on a chain of trust. Although route origin validation is beyond the scope of this SRA, this aspect will be discussed in detail with Cisco, particularly, considering the potential limitations of these chains of trust to cope with IP prefix hijacking.

Observe that the overlay must also guarantee that both the BGP advertisements as well as the overlay messages received from a neighbor domain are sent by trusted BGP routers and OEs (steps 18 and 19 in Algorithm. 1). Hence, each OE will store the set of authorized BGP routers and OEs in neighboring domains. Also note that we have considered an acknowledgment message (step 20 in Algorithm. 1), which is required for path authentication purposes. The ACK indicates that the other domains in the overlay have been notified about the existence of a new domain X in the network and that they have updated their PSGs accordingly. After the ACK, domain X can start sending BGP messages with the certainty that they will not filtered during the path-authentication process run by domains upstream.

As described above, after the initial exchange of information, only incremental updates are sent through the ePSP sessions. To illustrate the exchange of incremental updates consider the examples shown in Figures 13(a) and 13(c). In the first case, a single-homed customer C1 looses connectivity to the network, so its provider P1 advertises the change upstream (see Figure 13(b)) and the process continues through the primary branches of the overlay until all the ASs in the network are notified. In the second case, a multi-homed customer C2 looses connectivity with its primary provider P2, so both of them advertise the change (see Figure 13(d)) through the primary branches until all the ASs in the network are aware of the change. Observe that now C2 needs to restore its primary connection using its secondary session. In both examples, the convergence is entirely handled by the overlay layer (i.e., without triggering a single BGP update from the endpoints of the failure).

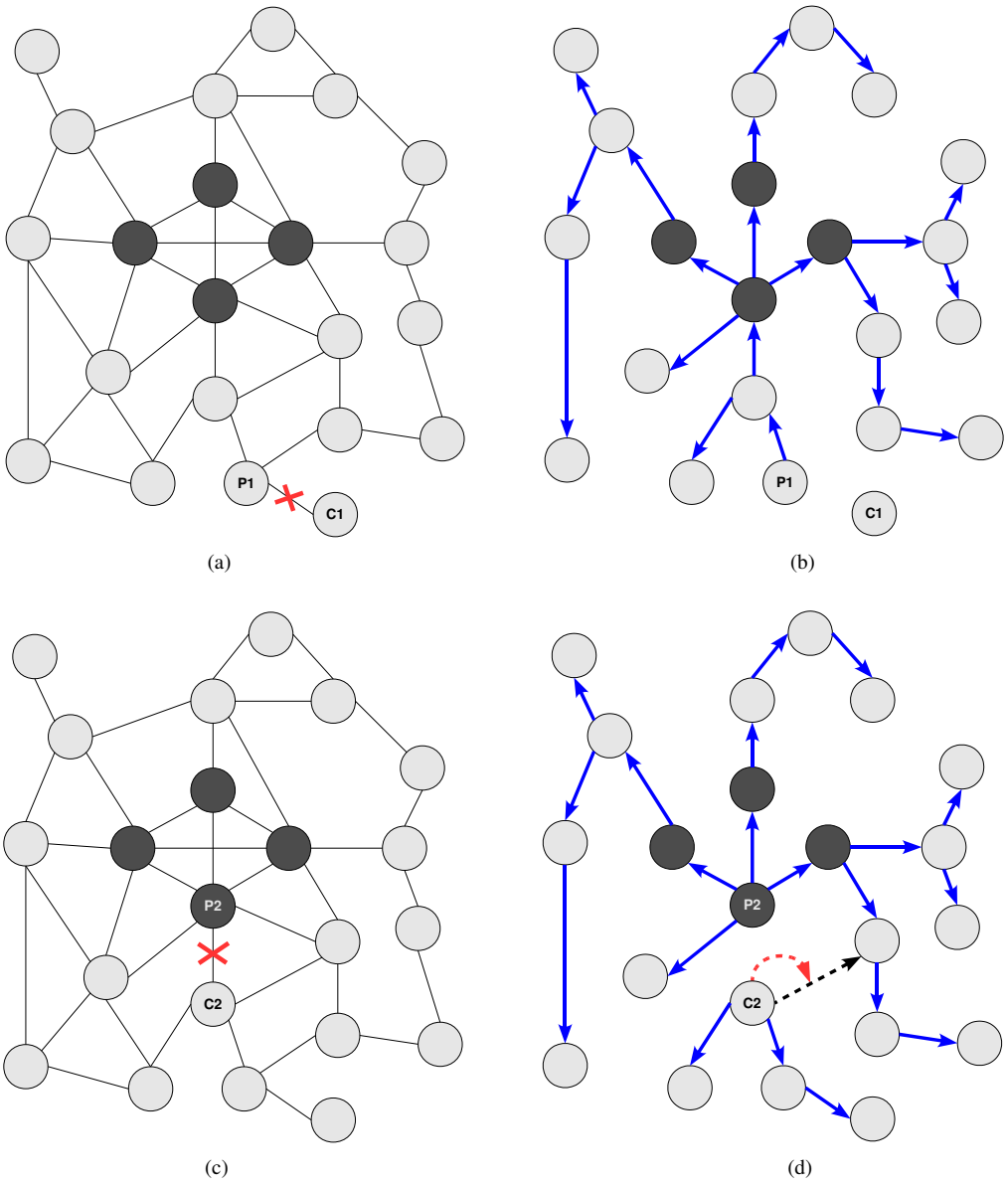


Figure 13: (a) The connectivity between a single-homed customer C1 and a provider P1 fails; (b) control messages through the overlay in order to fastly update the PSGs and avoid the useless BGP path hunting process; (c) now the connectivity between a multi-homed customer C2 and a provider P2 fails affecting its primary connection to the overlay; (d) control messages through the overlay in order to update the PSGs and avoid the BGP path hunting process. Note that now C2 needs to restore its primary session using the secondary one.

4.3 Bootstrap Processes

The start-up procedure of the overlay is separated into three different parts, first there is the Overlay Entity initialization, second there is the BGP router overlay registration part, and third the registration of other neighboring OE. In Figure 14 we sum-

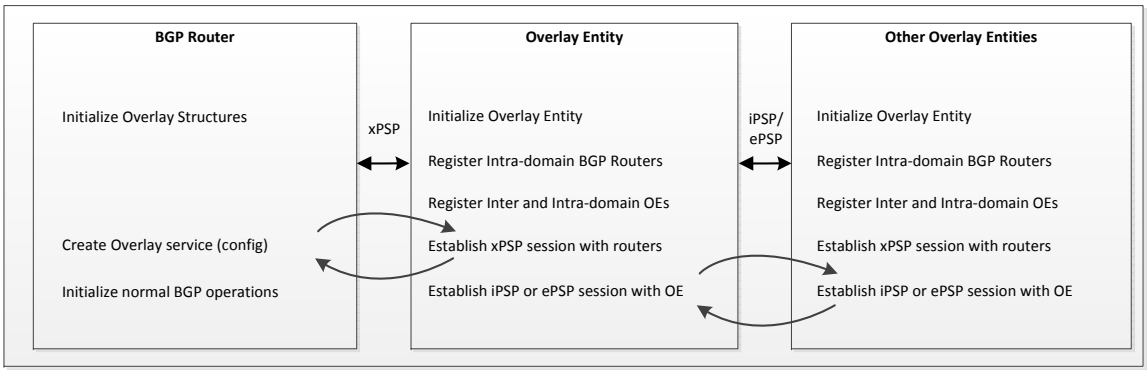


Figure 14: General boot process for both the OE and the BGP router.

marize this procedure, where we detail the different steps to start the service and the protocols used to supply the communication between the different entities.

As it can be noted, after the first initializations, the overlay entities start listening for a connection from the BGP routers, while at the same time they also listen for connections from other overlay entities. We proceed to describe in more detail these interactions, and defer the discussion about the protocols to Sections 5 and 6.

BGP Router Initialization: The BGP router must initiate the internal PSP status and register to the OE. This should be done before starting the normal operations of BGP, to avoid the potential loss of messages caused by discard actions performed by other OEs in the network. After this initialization, the router can start with the regular BGP initialization and establish the configured BGP sessions.

Overlay Entity Initialization: The overlay entity starts listening for registrations both from the BGP router and from neighboring OEs. During this process (and depending on the configuration of each OE), the OE will establish sessions with its neighbors as described in Sections 4.1 and 4.2. Even when it is not the goal of this initial prototype to implement all the security features outlined above, we lay the ground for an upgradeable system in terms of security. In particular, during the boot process each overlay entity administratively loads the list of legitimate BGP routers and neighboring OEs that are authorized to connect to the actual OE. This list can be updated by means of a configuration file during the boot process, or managed later from the *"Management Console"* as described in Section 4.4.

4.4 Overlay Entity Management Console

The list of valid commands is separated into two different groups: *configuration* commands and *informational* commands.

4.4.1 Configuration Commands

This set of commands are issued through the use of a configuration file or by entering in configuration mode (issuing the command **configure terminal**) from the *Man-*

agement Console.

- **psp entity** $<$ *oe_id* $>$: starts the PSP operations on the OE. For the initial tests we will use single node abstraction, and the OE identifier will be chosen as the AS number where the OE is running.
- **psp router-id** $<$ *oe_addr* $>$: defines the IP address that will be used in order to establish sessions with other OEs.
- **psp bgp register** $<$ *bgp_node* $>$: enables the system to accept as legitimate any connection attempt from the *bgp_node*.
- **psp neighbor** $<$ *neighbor_addr* $>$ **remote_as** $<$ *remote_as* $>$ **type** $<$ *neighbor_rel* $>$: enables the system to accept as legitimate a neighbor OE identified by the IP address *neighbor_addr* which manages the AS *remote_as*, with a relation *neighbor_rel* that can be Customer, Provider, or Peer.
- **psp mode** $<$ *oe_mode* $>$: establishes whether the current OE acts within a Tier-1 or not. This will determine the particular behavior in terms of message exchange within the overlay with the peers present in the configuration. Valid values for *oe_mode* are *tier1* or *standard*.
- **psp set primary** $<$ *oe_neighbor*, $\ldots$ $>$: sets the list of *oe_neighbor* as primary sessions for this domain.
- **psp enable security**[4]: adds support for security extensions. That is, besides the inspection of BGP messages and the eradication of the path hunting process, it adds to the protocol: prefix validation, next-hop attestation, and integrity tests.
- **psp shutdown**: used to disable the PSP protocol.

4.4.2 Informational Commands

The following list of commands is used in order to acquire internal information about the status of the overlay. All the actions are performed by prepending the word **show** to the command.

- **psp pst**: Outputs the current contents of the Path-State Table (PST).
- **psp information**: Reports information about the status of the OE.
- **psp neighbor** $<$ *oe_id* $>$: Informs about message statistics and status information of a particular neighbor.
- **psp neighbors**: Informs about message statistics and status information of all the neighbors.
- **psp bgp** $<$ *bgp_id* $>$: Informs about message statistics and status information exchanged with a particular BGP router running the `libPSP`.
- **psp primary**: Reports the list of primary OEs currently registered to the system.

[4]The implementation of these features is left for future versions.

5 Protocol Specification – xPSP

After describing the main entities present in the overlay, we now specify the mechanisms used to spread control information among the BGP routers and the OEs. To this end, we detail the specification of the *xPSP* (Cross-layer Path-State Protocol), which supports the interactions between a BGP router and its corresponding OE. We defer to Section 6 the specification of *ePSP* part of the protocol.

5.1 Message Format

To simplify the protocol implementation, all the messages sent by xPSP between BGP routers and the OEs have the same format. This is detailed in Figure 15, where:

- *Version:* (4 bits) refers to the protocol version used for the message exchange and currently is set to 1.
- *OP Code:* (4 bits) stands for the operation performed by the message. We distinguish different operations depending on the operation codes (see Table 2). Their meaning will be detailed later in this section.
- *Size:* (16 bits) is the size of the payload.
- *Flags:* (8 bits) this field contains different flags describing the content of the packet. At this stage, only the Most Significant Bit (MSB) is used, which is defined as the `T-flag` (type) indicating whether the packet corresponds to the xPSP or the ePSP protocol.
- *Sequence Number:* (32 bit) indicating a monotonically increasing number to avoid out of order packet injection. This sequence number is randomly initialized when an xPSP session is started between the BGP router and the OE, and it is increased on each xPSP packet exchanged between them.
- *Transaction ID:* (32 bit) is a random number that identifies the transaction being carried on. The Transaction ID is generated by the message originator and operates an identifier of the specific operation. This permits to have several ongoing transactions, e.g., legitimacy assessment of several BGP update messages at the same time. All the responses to the transaction will set the Transaction ID field to this particular value.

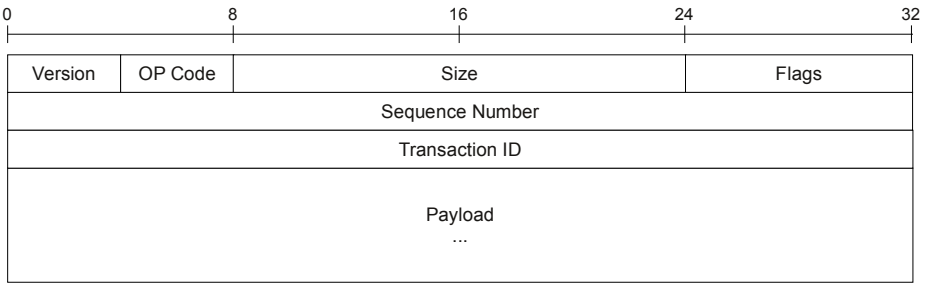


Figure 15: Message format both for the xPSP the ePSP protocol.

- *Payload:* (Variable size) refers to the parameters of the ongoing operation. The contents of the payload depend on the type of message transmitted:
  - *BGP router Register/Unregister* (OP Codes 0x00 and 0x01): the payload when using these OP Codes contains the IP address of the BGP router being registered/unregistered, while the response indicates the success or failure of the operation.
  - *BGP update* (OP Code 0x02): the message contains the whole payload of the original BGP message, and the response indicates the return value determining the result of the query, we further elaborate on this message later in this section.
  - *BGP link up/down* (OP Codes 0x04 and 0x05): the payload contains the connected/disconnected BGP neighbor, while the response indicates the success or failure of the operation.
  - *OE Status* (OP Code 0x08): when the BGP router queries for the OE status, the response to this message is basically a string describing the current status of the OE.

In the next subsection, we detail the different operations provided by the xPSP protocol and how the different messages interact in the system. At this point, it is important to recall that the control information managed by the OEs is decoupled from the information distributed by BGP. Consequently, all the topological and peering information in the overlay is obtained directly from the overlay and not from BGP.

5.2 Description of the Operations (OP Codes) in xPSP

Following the above message format, we now detail the information flow and the messages exchanged for each possible operation in the xPSP protocol.

5.2.1 Register and Unregister BGP Routers

With this message the BGP router may register or unregister to an already existing Overlay Entity. Due to security reasons, the OE keeps a closed list of legitimate BGP routers which can register to the actual OE within the domain. Once the BGP router is registered, the OE will answer with another register message. In case of error or any problem on an active session, it will return an unregister BGP router message and close the session. The main cause for rejection during the registration process is that the router is not present in the list of legitimate BGP routers that can register in the OE.

Value	*Description*
0x00	BGP router Register
0x01	BGP router Unregister
0x02	BGP Update
0x04	BGP Link Up
0x05	BGP Link Down
0x08	OE Status

Table 2: Operation codes for the xPSP messages.

5.2.2 BGP Update

When a new BGP update arrives to (or is generated by) the BGP router, it is immediately forwarded as an xPSP message to the OE. This message carries in its payload the whole BGP update with two purposes: i) to verify the legitimacy of the BGP message (in our case this means that the message has been generated by an authorized BGP peer; it contains legitimate IP prefixes; and the path is legitimate according to the Path-State Graph); ii) to eliminate the path explorations caused by BGP withdrawals.

After processing, the OE returns another xPSP message with the same Transaction ID, and with return value:

- `OV_DISCARD`: Discard the update as it is not legitimate.
- `OV_OK_NORMAL`: the BGP message is legitimate and BGP can continue its normal operations (the original BGP message goes in the payload of the xPSP reply).
- `OV_OK_NO_ACTION`: no further processing is required from BGP. This means that the overlay has already taken care of the processing, e.g., all the domains in the network have been notified about a topological change and the BGP FIBs/RIBs have been updated accordingly.
- `OV_OK_UPDATE_TABLES`: legitimate BGP message that requires the update of the FIB/RIB tables. The overlay instructs BGP about the change, which will be the one ultimately in charge of enforcing the update of the tables.

5.2.3 Local Link Up/Down

To fix ideas about the operation in this case let us consider the following example. The unreachability of IP prefixes is quite often caused by failures, some of which affect the PSGs. When this occurs, the overlay must avoid the generation of BGP messages from the router that is the origin of the failure. The workflow $1b \rightarrow 2 \rightarrow 3a \rightarrow 4a \rightarrow 5b$ in Figure 9 shows one such example. In that case, the xPSP message from the BGP router to the OE will contain the IP of the BGP Next-Hop that failed, while the return message from the overlay will be an `OV_OK_UPDATE_TABLES` as described in Section 3.

5.2.4 OE Status

When the BGP router administrator queries for the OE status, the OE returns basic information and statistics about its status.

6 Protocol Specification – ePSP

We now proceed to specify the external part of the Path-State Protocol (ePSP). As described in previous sections, this protocol enables the establishment of the primary and secondary connections between the OEs, and it is used among these latter for the dissemination of control information.

6.1 Message Format

The messages used for the exchange of information between the OEs have actually the same format that the ones specified for xPSP (see Figure 15), where the fields now indicate:

- *Version:* (4 bits) refers to the protocol version used for the message exchange and like in xPSP is currently set to 1.
- *OP Code:* (4 bits) As in the case of xPSP, this field indicates the operation performed by the message. We distinguish different operations depending on the operation codes (see Table 3). Their meaning will be detailed later in this section.
- *Size:* (16 bits) is the size of the payload.
- *Flags:* (8 bits) In the current version of the protocol we only define four different flags, the `T-flag` already defined in the previous section, and two other `scope-flags`, which determine the scope of the message to be sent when using the ePSP protocol:
  - `OV_F_CUSTOMER_CAST`: indicates that the message must be sent to a set of customers (see the peer-peer example in Figure 12).
  - `OV_F_PROPAGATE`: indicates that the message must be sent using the primary sessions, i.e., the one with the provider as well as the ones with the primary customers.

    The rest of the bits available are reserved for future use. An additional scope-flag is used in iPSP, namely, `OV_F_PROVIDER_CAST`, which indicates that the message must be sent to a set of providers. The detailed explanation about the usage of these flags in ePSP can be found below in this section.
- *Sequence Number:* (32 bit) it is a monotonically increasing number to avoid injection of out of order packets. Its role is the same as the one described for the xPSP protocol.
- *Transaction ID:* (32 bit) it is a random number identifying the transaction being carried on. Likewise, its role is the same as in xPSP.
- *Payload:* (Variable size) refers to the parameters of the operation. The contents of the payload depend on the type of message transmitted:
  - *Register/Unregister OE neighbor* (OP Codes 0x00 and 0x01): at this stage, when registering an OE, the content of the payload is the IP address of the source OE.
  - *Topology change – link insertion/deletion* (OP Codes 0x04 and 0x05): when informing about topological changes, the content of the payload consists of the list of AS pairs affected by the insertion/deletion of the links.
  - *New/Delete prefix for AS* (OP Codes 0x08 and 0x09): in this case, the content of the payload consists of the list of prefixes.
  - *New/Delete BGP next-hop* (OP Codes 0x0a and 0x0b): similar to the previous case; the payload contains the list of authorized BGP next-hops.

  It is important to notice that, once the security features are enabled in the overlay, the information conveyed must also contain the digital signature for the payload in order to guarantee the integrity and authenticity of the message.

Value	*Description*
0x00	OE register neighbor
0x01	OE unregister neighbor
0x04	Topology change – link insertion
0x05	Topology change – link deletion
0x08	New IP prefix for AS
0x09	Delete IP prefix from AS
0x0a	New next BGP hop
0x0b	Delete next BGP hop

Table 3: Operation codes for the ePSP messages.

6.2 Description of the Operations (OP Codes) with ePSP

Following the above message format, we now detail the information flow and the messages exchanged for the operations provided by the ePSP protocol.

6.2.1 OE Node Registering/Unregistering

When an OE joins the overlay, it sends an OE register message to its neighbors and establishes primary and secondary sessions as described above. Due to security reasons, the list of neighbors for each OE will be administratively configured within the node. Once an OE receives a register message, it will answer with another register message, and, in case of error or any problem on an active session, it will send an unregister neighbor message and close the ePSP session. The OE that receives the register/unregister message will inform about the topological change to other domains by following the rationale in Section 6.2.2 (note that registration/unregistration of an OE will often affect the PSGs of its neighbors).

6.2.2 Topology Change – Link Insertion and Deletion

The payload of these messages will contain the list of links inserted to or deleted from the network. When an OE receives such message, it will acknowledge with another message with an empty payload containing the same *OP Code* and *Transaction ID* received. It is important to notice that in the case of a Customer-Provider link, both OEs detecting the event in the link will inform their primary customers, and the one in the provider will also inform upstream about the change through its primary session. To this end, the message sent will have the `OV_F_PROPAGATE` flag set, whereas the `OV_F_CUSTOMER_CAST` flag will be unset. If, on the other hand, the event refers to a Peer-Peer link, then, as discussed in Section 4.1, the OEs must only advertise the change to their customers. In such a case, one option is to set the flag `OV_F_CUSTOMER_CAST`, in order to instruct the customers that they must propagate the message to their own customers until all customers downstream are notified.

6.2.3 New/Remove AS IP prefix

When an AS acquires new IP prefixes, the OEs in the AS must advertise the prefixes to their neighbors—if not, any BGP update that contains a new IP prefix will be systematically discarded by neighboring OEs. In our initial prototype, the ePSP messages sent from the originating AS will contain the list of new prefixes and they will be

distributed only to adjacent domains, so none of the flags will be set in these messages. With this scheme, messages containing IP prefixes originated by a neighbor could be validated and processed locally, whereas the legitimacy of messages containing prefixes originated by a distant domain would depend on previous attestations (chain of trust). For the moment, these features are in fact out of the scope of this SRA, but as described in Section 4.2 they shall be discussed in detail with Cisco.

6.2.4 New/Delete BGP next-hop

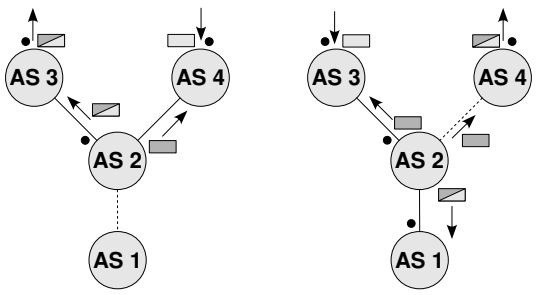
The overlay must be able to assess the validity of the source of a BGP message, so each OE stores a list of authorized BGP speakers from its neighbors. Any change in this regard must be notified and validated through the ePSP session. To this end, the message payload will contain the list of border routers that must be added (or the ones that must be removed).

6.3 Examples of the Interaction between ePSP and BGP

For the sake of clarity, we illustrate a few cases where the AS-level topology changes, i.e., when an AS or an interdomain adjacency is inserted to, or deleted from, the AS graph. More specifically, we illustrate the interactions for events that affect the PSGs, but as described in Section 3, the application of the overlay and hence the interactions between BGP and ePSP are not limited to such events.

6.3.1 A New Customer-Provider Adjacency is Created

Figure 16 shows two different settings for this case as well as the flow of some of the ePSP and BGP messages exchanged: (a) when a new AS appears in the network (single-homed case; see AS1 in Figure 16(a)), and (b) when an already connected AS establishes a new adjacency with another AS (multi-homed case; see AS2 in Figure 16(b)). In the remainder of this section, the dotted lines in the figures shall refer to the adjacency involved (i.e., a new adjacency is added or an existing one is removed), while the solid lines correspond to already existing and stable adjacencies. The black dots contiguous to the AS nodes indicate the links used by the latter as their primary session in the overlay. The light gray boxes indicate the flow of ePSP messages, the dark gray ones stand for regular BGP updates, and finally the mixed ones indicate the flow of both ePSP and BGP updates.



(a) Single-homed case (b) Multi-homed case

Figure 16: A new customer-provider adjacency is created.

Overlay actions: The new adjacency needs to be disseminated through the overlay, since it affects the PSGs. In the single-homed case, the provider AS2 will use its primary connections to send the ePSP updates and will also send its topological view to AS1. Note that AS4 will receive the update through its primary session. In the multi-homed scenario, the provider AS4 will use its primary connections to send the updates, while AS2 will send updates only to its primary customers. In this case, it is AS3 the one that receives the update through its primary session to the overlay.

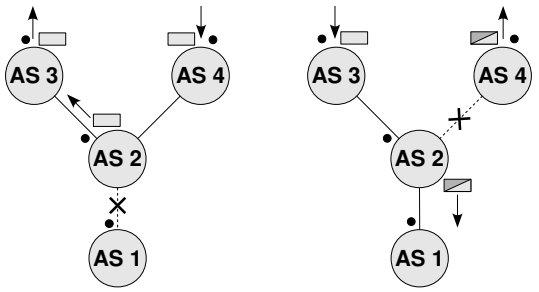
BGP actions: Once the topological change has been disseminated and the overlay is prepared, the ASs that were connected through the new adjacency can now establish a BGP session and start sending BGP updates as usual.

6.3.2 A Customer-Provider Adjacency is Removed

Figure 17 shows the case when an adjacency is removed (e.g., an interdomain link fails, a customer switches to another provider, etc.). As in the previous case, we analyze both the single-homed and the multi-homed scenarios.

Overlay actions – Single-homed case: Consider the example shown in Figure 17(a), where all the prefixes originated by AS1 become unreachable after the failure of the link between AS1 and AS2. When this occurs, the overlay entity detecting the failure in AS2 takes control and returns an `OV_OK_UPDATE_TABLES` message to the BGP router through the xPSP protocol. Note that the topological change is disseminated through the primary sessions, where each OE receiving the message will update its PSG and send an `OV_OK_UPDATE_TABLES` message in order to remove the corresponding information from the FIB/RIB tables of the BGP router that it manages. Similar settings will be used with our prototype, to demonstrate that the ePSP protocol can not only remove the path exploration from the routing system but can also dramatically reduce the churn of updates during a routing convergence.

BGP actions – Single-homed case: All the BGP updates triggered to inform about the event are processed and discarded by the source OE in AS2. In this sense, the overlay shields the rest of the routing system, since these BGP messages will never be sent from the router where the failure occurs (though its FIB and RIB will be updated accordingly).



(a) Single-homed case (b) Multi-homed case

Figure 17: A customer-provider adjacency is removed.

Overlay actions – Multi-homed case: As in the single-homed case, the ePSP protocol will quickly disseminate the topological change through the primary trees in the overlay. As the OEs are notified, they will send an `OV_OK_UPDATE_TABLES` message to their BGP routers, in order to remove from the FIB/RIB the entries affected by the failure.

BGP actions – Multi-homed case: Note that while the withdrawal of unavailable routes is now controlled by the overlay, the routers may still require BGP updates in order to advertise their new best routes (see Figure 17(b)).

The interactions involving peer-peer adjacencies can be illustrated by extending the example shown in Figure 12 and by following the rationale described above.

At this stage, we are focused on prototyping both the xPSP and the ePSP parts of the protocol. For the testing of the interactions between the overlay and BGP, we shall maintain the single node abstraction, so the internal iPSP is left for future work and will not be tested in this prototype.

A Appendix—Graphs on Path Vectors: Theory

1. Introduction

Over the last decade, path vector protocols have aroused controversy, questioning their capacity to provide the functionality expected from the interdomain routing system. On the one hand, these protocols have some undeniable strengths, such as the capability of distributing reachability information and providing loop-free paths among domains in a straightforward and operationally scalable way. On the other hand, multiple studies show that the main path vector protocol, the Border Gateway Protocol (BGP) [10], suffers from several problems (see, e.g., [11]), including, slow convergence [12], high churn rate of route advertisements [13, 14], limited means for attaining Traffic Engineering (TE) objectives [15], and security vulnerabilities [16, 17].

On the contrary, link-state protocols can substantially mitigate these problems, but their characteristics render them inappropriate for handling the interdomain routing system. This is because the link-state paradigm was not conceived for managing the policies that control the commercial relationships and the routing among domains. Besides, the "link state" is not necessarily relevant at the interdomain level; what matters today in terms of routing information is the Autonomous System (AS)-path state, which we refer to here as the *path-state*.

In light of these constraints, alternatives have been sought so as to integrate the strengths of link-state routing seamlessly into the interdomain routing system—for arguments in support of this integration see, e.g., [1], [2]. These efforts, however, have not succeeded as yet. A plausible reason for this is that, the adoption of a routing paradigm that combines the strengths of path vectors and link state-like routing poses a twofold challenge.

The Theoretical Challenge—The core strengths of link-state protocols are rooted in the distributed computation and maintenance of a graph of the network. As we shall show, from a computer science perspective, the challenge is that link state-like graphs cannot offer a consistent view of the forwarding paths in an internetwork subject to routing policies.

The Practical Challenge—From a practical viewpoint, the key challenge is to amalgamate the two different routing styles in a way that requires neither the wholesale replacement of BGP nor the development of upward-compatible extensions, making BGP implementations more complex than they are today. Even though the need for replacing BGP is perfectly defensible from a theoretical point of view, the practical implications of its replacement have thus far hindered any initiative in this direction.

In this two-part paper, we present a new and promising approach with the potential to solve both challenges at once. Part I addresses the theoretical challenge, whilst Part II tackles the practical issues and explores some possible applications of the solution developed in Part I.

Our main contributions can be summarized as follows. In Part I, we expose in detail the theoretical challenge, and propose a graph representation through which domains can keep consistent views of the forwarding paths and domain-level connectivity of an internetwork subject to routing policies. We call this graph abstraction a *Path-State Graph* (PSG). As we shall show, PSGs respect the opaqueness of providers, since they are constructed with the routing information resulting from the enforcement of the usual export policies between routing domains. In Part II, and following the approach

proposed by White in [1], we discuss how to overlay the PSGs over the BGP protocol, and describe how to exploit these graphs in different ways, e.g., to dramatically reduce the time and churn rate of an interdomain routing convergence. In particular, we show how PSGs can be applied for *removing* the path-exploration from the Internet's routing system; and more importantly, we show how to achieve this without replacing nor extending BGP.

It is worth highlighting that this two-part paper does not intend to provide a closed-form solution for the challenges outlined above, but rather to offer a starting point for discussion and consideration by the Internet community.

The remainder of Part I is structured as follows. In Section A, we provide the necessary background and introduce the routing policies used in the rest of the paper. Section A presents the problem of the different topological views of domains, while Section A analyzes the inconsistencies associated with the paths on the graphs inferred by domains. The PSGs are introduced in Section A, and their main properties are studied in Section A. In Section A, we review related work. Finally, Section A concludes the paper, and outlines potential applications of the PSGs, some of which are explored in detail in Part II.

2. Preliminaries

As shown in Figure 18a, an internetwork is usually represented as an undirected AS graph $G(V,E)$, where each vertex $v \in V$ represents a domain, and the edges $e \in E$ represent the interconnection between neighboring domains. This graphical abstraction does not necessarily reflect the physical connectivity of the network, but offers a reasonable basis for studying the properties of interdomain routing algorithms and protocols.

In practice, this routing is controlled by the commercial relationships between domains. The large majority of these relationships can be classified into two categories: *customer-provider* and *peer-peer*. The former occurs when a domain purchases connectivity and pays for transit to another domain. The latter applies when two domains that exchange a significant amount of traffic, agree to connect directly to each other to avoid transiting through (and thus pay) a third party provider. Peering domains share the costs of the connection between them, so there is no customer/provider relationship in the peer-peer case.

Strictly speaking, there is a third category called *sibling-sibling*, but it is quite infrequent—these peering relationships are sometimes used between merging companies. According to data from CAIDA's AS Relationships Dataset [18], less than 0.3% of the total number of relationships between Internet domains were siblings in March of 2010. Since the large majority of the relationships between domains falls into the first two categories, as in [19], sibling relationships are not considered in this paper.

In this framework, besides the interconnection of domains, each edge in the AS graph in Figure 18a represents either a customer-provider (CP) or a peer-peer (PP) relationship between its endpoints. Figure 18b shows the AS graph with a typical set of commercial relationships between its vertices. In this example, we assume that AS1 is a customer of AS2 and AS5, while AS4 is a customer of AS3 and AS5. In addition, we assume that AS2, AS3, and AS5, form a triangular peering relationship among them. As shown in Figure 18c, we will represent the flow from a provider to a customer as a directed edge $\overrightarrow{\text{PC}}$, and the flow in the opposite direction as a directed edge $\overrightarrow{\text{CP}}$. Likewise, the flow toward a peer will be represented as a directed edge $\overrightarrow{\text{PP}}$.

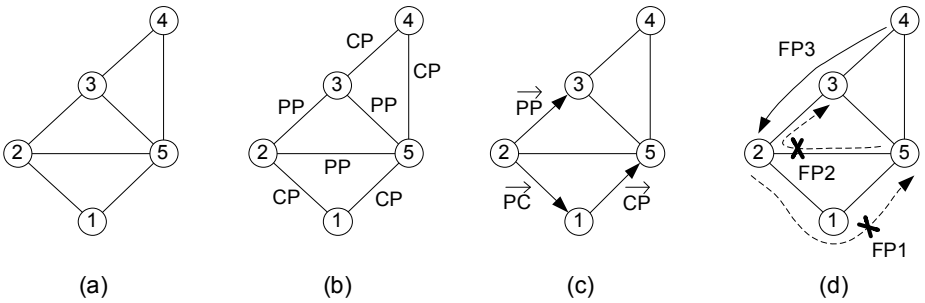


Figure 18: (a) An AS graph composed of five domains; (b) adding commercial relationships between domains; (c) directed edges: provider → customer ($\overrightarrow{\text{PC}}$), customer → provider ($\overrightarrow{\text{CP}}$), and peer → peer ($\overrightarrow{\text{PP}}$); (d) examples of valley-free Filtering Polices FP1–FP3.

2.1. Valley-free Routing

At present, the commercial relationships described above entail a set of routing policies among Internet domains—known as *valley-free* routing policies [20]—which derive in the following Filtering Policies (FPs) of a domain.

FP1) Traffic coming from a provider will not be forwarded to a peer or another provider. In other words, a $\overrightarrow{\text{PC}}$ edge can be followed neither by a $\overrightarrow{\text{PP}}$ nor a $\overrightarrow{\text{CP}}$ adjacent edge.

FP2) Traffic coming from a peer will not be forwarded to another peer or provider, i.e., a $\overrightarrow{\text{PP}}$ edge can be followed neither by a $\overrightarrow{\text{PP}}$ nor a $\overrightarrow{\text{CP}}$ adjacent edge.

FP3) Traffic coming from or directed to a customer can always be forwarded by a provider.

Table 4 summarizes the transits that are allowed and forbidden between two adjacent edges, (w,u) and (u,v), due to the filtering policies FP1–FP3. These filtering rules restrict the set of paths available in an internetwork. For instance, Figure 18d shows that a customer, such as AS1, will never provide transit between its providers, AS2 and AS5. This is achieved by the enforcement of filtering policy FP1 in AS1. In addition,

Table 4: Filtering policies (FP1–FP3) applied by domain u for the transit from domain w to domain v through u: $w \rightarrow u \rightarrow v$.

		Adjacent edge (u,v)		
		$\overrightarrow{\text{CP}}$	$\overrightarrow{\text{PC}}$	$\overrightarrow{\text{PP}}$
First edge (w,u)	$\overrightarrow{\text{CP}}$	✓	✓	✓
	$\overrightarrow{\text{PC}}$	×	✓	×
	$\overrightarrow{\text{PP}}$	×	✓	×

AS2 will not provide transit between its peers, AS5 and AS3, which is enforced by filtering policy FP2 in AS2. On the other hand, according to filtering policy FP3, AS3 will always forward packets originated at AS4. In what follows, we assume that the routing relationships between the vertices of the AS graphs analyzed in this paper are subject to the filtering policies described above.

2.2. Valley-Free Routes: the Import and Export Model

The routes available in an internetwork are determined by the import and export policies of routing domains. For each domain u, these policies can be split into two groups. The first group consists of the application of filtering policies FP1–FP3, which ensure that the routing across domain u is valley-free. Once u guarantees that the transit through its domain is valley-free, it can apply another group of policies to manage its traffic distribution, both inbound through its export policies and outbound through its import policies. The combination of filtering policies FP1–FP3, along with the traffic distribution policies independently applied by domains, is what ultimately determines the routing information available at each domain—thus conditioning the graphs that can be inferred by these latter.

In this paper, we assume that the routes are composed of path vectors. It is important to highlight that, differently from BGP-based routing, where domains only export their best route for a destination, our interest is to analyze the graph representations that can be inferred by domains, in a network that is subject to the filtering policies FP1–FP3 and the traffic distribution policies locally applied by domains. Therefore, in the notation introduced next, we assume that, in principle, a domain can import and export all the routes compatible with the filtering rules FP1–FP3, and its local traffic distribution policies. In other words, the graph constructions and the path availability analysis that will be carried out in Sections A, A, and A, shall be made considering the combination of valley-free and local routing policies, independently of the routing protocol that implements them—as long as the routes exchanged between domains preserve the path vector format. As a result, the only possible pruning of the graphs inferred by domains will be due to the routing policies.

As shown in Figure 18, we will focus on the construction of graphs of interconnectivity at the AS-level, so the path availability analysis shall be made between the vertices of these AS graphs. In Section A, we discuss the challenges entailed in constructing graphs subject to routing policies, where the granularity of the routing destinations is kept at the IP prefix level.[5]

Under these assumptions, let u, v, and w, represent three domains, such that u is a neighbor of both v and w. Let $X_v(u)$ denote the set of routes exported from v to u, and $O_u^v(r)$ a binary function on route r, such that $O_u^v(r) = 1 \Leftrightarrow$ route r matches u's outbound traffic policy through v. As mentioned above, in order to study the graphs that can be inferred by domains under routing policies, we assume that the subset of routes from $X_v(u)$ kept by u after applying its import policy is:

$$I_u(v) = \{r \in X_v(u) \ / \ u \not\subseteq r\} \setminus \{r,\ O_u^v(r) = 0\} \tag{1}$$

i.e., a domain u imports all the routes received from a neighbor v, except those that represent forwarding loops and those that do not match u's outbound traffic policy.

[5] In Part II of this paper, we show that keeping the granularity in the graph constructions at the AS-level is remarkably useful in practice, and it is sufficient to deal with some of the main problems of the BGP protocol.

The complete set of routes imported by u is denoted as I_u, and is given by:

$$I_u = \bigcup_{(u,v)} I_u(v) \tag{2}$$

As summarized in Table 4, the routes exported from u to w are subject to the valley-free routing model. Let $\mathcal{F}_{u,w}$ denote the filtering function that enforces policies FP1–FP3 in the routes advertised from u to w. Likewise, let $I_u^w(r)$ be a binary function on route r, such that $I_u^w(r) = 1 \Leftrightarrow$ route r matches u's inbound traffic policy for the traffic coming from w. Thus, the routes exported from u to w can be expressed as:

$$X_u(w) = \bigcup_{(u,v),\, v \neq w} u \circ r,\ r \in \mathcal{F}_{u,w}(I_u(v)) \setminus \{r,\ I_u^w(r) = 0\} \tag{3}$$

which basically consists in prepending u—operator $\circ$ in (3)—to all the routes r resulting from the filtering function $\mathcal{F}_{u,w}$, except those that do not match u's inbound traffic policy.

Equations (1)–(3) model the import and export policies used in this paper, which we shall refer to as *Valley-Free Routing* (VFR) policies. This simple model captures the routing policies currently used in the Internet, with the advantage that the availability of paths in the network can be studied independently of the protocol used for the distribution of path vectors.

2.3. Terminology and Definitions

Before proceeding and to avoid ambiguities, we define some relevant graph theory terms used throughout this paper [21].

Definition 1. *(Directed Walk) In an directed graph (or digraph) G, a directed walk from vertex v_0 to vertex v_n is a sequence $W = \langle v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n \rangle$ of vertices and edges, such that each edge e_i is directed from vertex v_{i-1} to vertex v_i, i.e., $tail(e_i) = v_{i-1}$ and $head(e_i) = v_i$.*

Definition 2. *(Concatenation) The concatenation of two directed walks $W_1 = \langle v_0, e_1, \ldots, e_k, v_k \rangle$ and $W_2 = \langle v_k, e_{k+1}, \ldots, e_n, v_n \rangle$, such that W_2 begins where W_1 ends, is a directed walk, and it is denoted as: $W_1 \circ W_2 = \langle v_0, e_1, \ldots, e_k, v_k, e_{k+1}, \ldots, e_n, v_n \rangle$.*

Definition 3. *(Trails on Digraphs) In a digraph G, a trail is a directed walk with no repeated edges.*

Definition 4. *(Paths on Digraphs) In a digraph G, a path is a trail with no repeated vertices (except possibly the initial and final vertices).*

3. Different topological views under Valley-free Routing Policies

We will now analyze how routing policies affect the potential construction of a link state-like graph of interconnectivity of the network. As in any link-state protocol, we focus on the distributed inference of the graph, i.e., each node (domain) in the network computes a graph based on the routing information that receives. Later, in Section

A, we analyze the case where domains can obtain directly the interconnectivity graph, assuming that the graph is precomputed by external means.

Consider again the example in Figure 18b, which is shown at the center of Figure 19. Henceforth, the undirected AS graph representing the interconnectivity of the network will be denoted as graph $G_0(V,E)$. To facilitate the interpretation of the graphs depicted in Figure 19, Table 5 details the routes resulting from the import and export VFR policies described in Section A. More precisely, each row of Table 5 contains all the valley-free routes available in one domain to the rest of domains in the network. Therefore, the i-th row of the table holds the information that domain v_i has available to infer the network graph. For simplicity in the exposition, in the example in Table 5, we assumed $O_u^v(r) = 1$, and $I_u^w(r) = 1 \; \forall \; r \subset G_0$ and $u, v, w \in V$.

Let $\mathcal{V}(G_0)$ represent the set of routing records resulting from the application of the VFR policies on G_0. A set of routing records, $\mathcal{V}(G_0)$, organized in the form of the matrix shown in Table 5, shall be referred to as the *Network Routing* (NR) table, and will be denoted as table $\mathcal{R} = \mathcal{V}(G_0)$. Hence, using a notation similar to that in [22], the routing system can be denoted as $\mathcal{S} = \langle G_0, \mathcal{R} \rangle$.[6]

It is worth noting that, due to the VFR policies, the paths available in one domain might not necessarily be known by other domains. For instance, rows 1 and 2 of the NR table show the paths seen by AS1 and AS2, respectively. Even though in the example both AS1 and AS2 are adjacent to AS5 (see Figure 19), the path [5,3,4] is available for AS1, but this path is not known by AS2 (cf. rows 1 and 2 of the NR table). This is due to the enforcement of filtering policy FP2 in AS5 for the transit AS2 $\xrightarrow{\text{PP}}$ AS5 $\xrightarrow{\text{PP}}$ AS3, since AS5 will not advertise the paths learned from AS3 to AS2. From the NR table, it is possible to observe that the paths in row 2 contain neither the sequence (3,5) nor (5,3), so from AS2's perspective, AS3 and AS5 are disconnected. In other words, AS2 cannot infer that the network physically admits the path [2,5,3,4], unless an external source of information is used.[7]

Figure 19 shows the graphs that can be inferred by providers AS2, AS3, and AS5, according to the valley-free paths they know (cf. rows 2, 3, and 5, respectively, of the NR table). These graph constructions are denoted as G_2, G_3, and G_5, and they are represented as digraphs, where the set of directed edges captures the paths available from each provider to any other domain in the network. The main observation is that, under VFR policies, the link state-like graphs inferred by the providers in the example are all different from each other.[8]

As described in Section A, the application of routing policies by domains may lead to asymmetries in the advertisement of paths through the network. More specifically, if v and w are neighbors of a domain u, the routes advertised from u in the direction $u \rightarrow v$ can differ from the ones advertised in the direction $w \leftarrow u$. These asymmetries explain the directed nature of the graphs that can be constructed by domains. Indeed, in the remainder of this section, we will prove in Theorem 1 that, the presence of a directed edge (v_k, v_{k+1}) in the graph inferred by a domain v_i, is not sufficient to ensure that the opposite edge, (v_{k+1}, v_k), must also be present in the graph. Theorem 1 and its

[6] Differently from [22], our focus at this stage is on the graph constructions that can be inferred by domains from the stationary state of the routing, so the evolution from an initial state s_0 is not considered in $\mathcal{S}$.

[7] In practice, external sources of information, such as route views [23] or CAIDA's ARK measurement infrastructure [18], can be used to estimate the set of paths available in other domains. Our focus here is on the inference of graphs from the information distributed within the routing system, so external means of information are not considered at this stage.

[8] The topological views of the customers AS1 and AS4 pose additional complexities, and will be analyzed in detail in Section A.

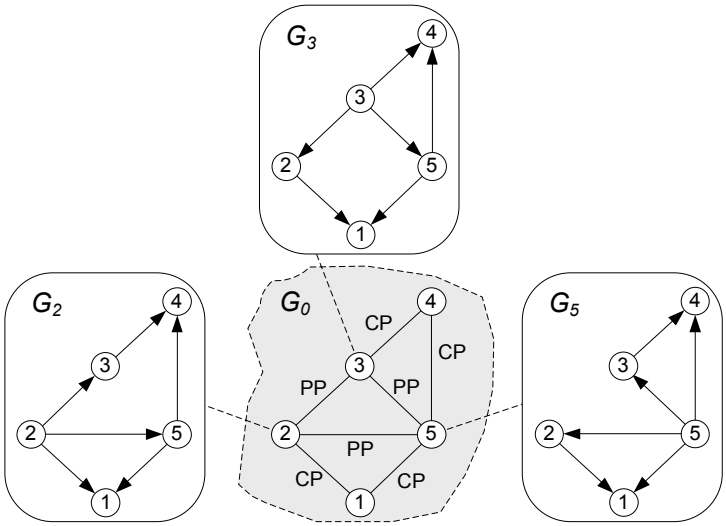


Figure 19: Due to the VFR policies, the providers AS2, AS3, and AS5, have different views of the AS graph G_0.

Table 5: The Network Routing (NR) table $\mathcal{R}$ the rows show the valley-free routes seen by each domain to the rest of domains in the network.

	1	2	3	4	5
1	-	[2] [5,2]	[2,3] [5,3]	[2,3,4] [5,4] [2,5,4] [5,3,4]	[5] [2,5]
2	[1] [5,1]	-	[3]	[3,4] [5,4]	[5]
3	[2,1] [5,1]	[2]	-	[4] [5,4]	[5]
4	[3,2,1] [5,1] [3,5,1] [5,2,1]	[3,2] [5,2]	[3] [5,3]	-	[5] [3,5]
5	[1] [2,1]	[2]	[3]	[4] [3,4]	-

corollary will be then used in Theorem 3, where we will prove that the graphs inferred by domains are all mutually different.

Theorem 1. *(Directedness of the graphs) Let v_k and v_{k+1} be two adjacent domains in a routing system $\mathcal{S}$ with $|V| > 2$ domains, such that the directed edge (v_{k+1}, v_k) is not contained in the path vectors available in domain v_i. The presence of the opposite edge (v_k, v_{k+1}) in the path vectors known by v_i is necessary, but not sufficient to include the edge (v_{k+1}, v_k) in the graph inferred by domain v_i.*

Proof. The necessity is clear, since if neither (v_k, v_{k+1}) nor (v_{k+1}, v_k) were contained in the path vectors known by v_i, then, from the perspective of v_i, v_k and v_{k+1} are disconnected. Thus, (v_{k+1}, v_k) will not be present in the graph inferred by v_i. To prove the insufficiency, we construct the example shown in Figure 20. Suppose that the vertices in the tuple $\{v_k, v_k{}', v_k{}''\}$ are all adjacent to v_{k+1}, where v_k and $v_k{}'$ are providers of v_{k+1}, whilst $v_k{}''$ is a peer. Accordingly, the routes advertised by v_{k+1} toward v_i are sent through v_k and $v_k{}'$, but not through $v_k{}''$, since the latter will not provide transit between the large internetwork and v_{k+1}. Assuming that the upstream domains until v_i is reached, honor v_{k+1}'s advertisements, then, v_i will receive at least two path vectors containing v_{k+1}, one through v_k, and another through $v_k{}'$. These path vectors are denoted as $\overrightarrow{p_1} = [\ldots, v_k, v_{k+1}, \ldots]$, and $\overrightarrow{p_2} = [\ldots, v_k{}', v_{k+1}, \ldots]$, respectively (see Figure 20a). Since the directed edges (v_k, v_{k+1}), and $(v_k{}', v_{k+1})$, are contained in the path vectors known by v_i, they will both be present in the graph inferred by v_i, which we denote as graph G_i. Indeed, $\{\overrightarrow{p_1}, \overrightarrow{p_2}\} \subset G_i$.

Consider now the opposite edges, (v_{k+1}, v_k), and $(v_{k+1}, v_k{}')$. Notice that the paths between any pair of vertices in the tuple $\{v_k, v_k{}', v_k{}''\}$ through v_{k+1} are not valley-free. This implies that v_{k+1} will not provide transit between the vertices in the tuple, so by construction (see Figure 20a), the directed edges, (v_{k+1}, v_k), and $(v_{k+1}, v_k{}')$, cannot be contained in the path vectors known by v_i. Therefore, both (v_{k+1}, v_k) and $(v_{k+1}, v_k{}')$ in the example satisfy the hypothesis of the Theorem.

The fact that v_i receives the paths $\overrightarrow{p_1}$ and $\overrightarrow{p_2}$, implies that, according to the filtering policies FP1–FP3, these paths are valley-free. By extending the transits in Table 4 to an arbitrary large chain of domains, it can be easily shown that if a path $\overrightarrow{p} = [v_0, v_1, \ldots, v_m]$ is valley-free, then its reverse path, $\overleftarrow{p} = [v_m, \ldots, v_1, v_0]$, is also valley-free. Hence, the reverse paths, $\overleftarrow{p_1} = [\ldots, v_{k+1}, v_k, \ldots]$, and $\overleftarrow{p_2} = [\ldots, v_{k+1}, v_k{}', \ldots]$ are both valley-free, and thus could be used by other domains to reach v_i. As a result, domain v_i may assume that the directed edges contained in $\overleftarrow{p_1}$ and $\overleftarrow{p_2}$ should also be present in its graph construction. Suppose now that v_i adds these edges to G_i, among

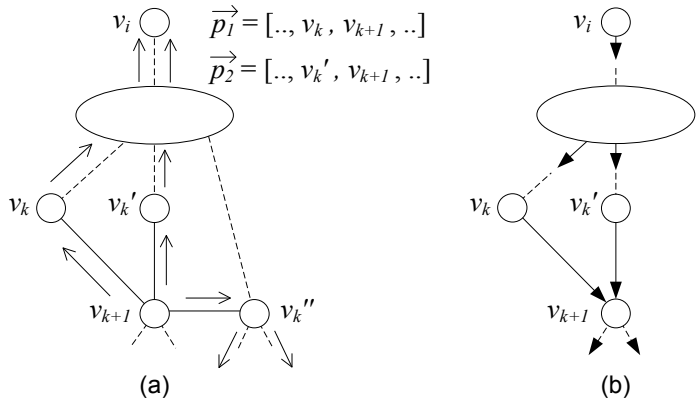


Figure 20: (a) The arrows represent the flow of advertisements sent from v_{k+1} according to the VFR policies; (b) graph inferred by domain v_i.

which are (v_{k+1}, v_k) and $(v_{k+1}, v_k{}')$ even though they are not contained in the path vectors known by v_i. Since $\overrightarrow{p_2} \subset G_i \Rightarrow \exists\ W' = \langle v_i, \ldots, v_k{}', (v_k{}', v_{k+1}), v_{k+1} \rangle \subset G_i$, a directed walk from v_i to v_{k+1} along $\overrightarrow{p_2}$, with neither repeated edges nor vertices. Let $W = \langle v_{k+1}, (v_{k+1}, v_k), v_k \rangle \subset G_i$ be the directed walk between v_{k+1} and v_k through (v_{k+1}, v_k). The concatenation of the two walks, $W' \circ W$, is a directed walk on G_i with neither repeated edges nor vertices, so $\overrightarrow{p_3} = W' \circ W$ is a valid path on graph G_i. This means that on G_i, v_i can reach v_k through $\overrightarrow{p_3} = [\ldots, v_k{}', v_{k+1}, v_k]$, which contradicts the fact that v_{k+1} does not provide transit between the vertices in the tuple $\{v_k, v_k{}', v_k{}''\}$. Thus, the directed edge (v_{k+1}, v_k) must not be present in G_i. Hence, the presence of a directed edge in the path vectors known by a domain is not sufficient to include the opposite edge in the graph inferred by the domain. This completes the proof. □

In Figure 20b, the absence of the directed edge $(v_k{}'', v_{k+1})$ in the path vectors known by v_i illustrates the necessity in Theorem 1, while Figures 20a and 20b together illustrate the insufficiency.

From Theorem 1, we conclude that the addition of the directed edge (v_{k+1}, v_k) to G_i produces new paths on G_i, some of which may violate the valley-free routing principle. This is the main reason for not including directed edges in G_i that are not contained in the path vectors known by v_i. A secondary reason is that such inclusions may also lead to inconsistencies in the traffic distribution policies of the domains in G_i. More specifically, the presence of a path vector $\overrightarrow{p}$ in the paths known by v_i does not necessarily imply that the reverse path $\overleftarrow{p}$ will be used in the network, since its use depends on the policies of other domains. The autonomic nature of routing policies allows the occurrence of these asymmetries, so a domain cannot include a directed edge in its graph construction unless that the directed edge is part of one of the path vectors stored in its corresponding row of the NR table. We formalize this in the following corollary.

Corollary 2. *Let v_i be a domain in a routing system $\mathcal{S}$ with $|V| > 2$ domains, and $\mathcal{R}_i$, $1 \leq i \leq |V|$, be the corresponding row of v_i in table $\mathcal{R}$. Let $G_i = G_i(V_i, E_i)$ denote the graph that can be inferred by v_i from the path vectors contained in $\mathcal{R}_i$. The directed edge $(v_k, v_{k+1}) \in E_i$ if and only if there exists a path vector $\overrightarrow{p}$ in $\mathcal{R}_i$, such that $(v_k, v_{k+1}) \subseteq \overrightarrow{p}$.*

Proof. (Necessity) Suppose toward a contradiction that $(v_k, v_{k+1}) \nsubseteq \overrightarrow{p}$, $\forall\ \overrightarrow{p}$ in $\mathcal{R}_i$. Hence, (v_k, v_{k+1}) is not contained in the path vectors known by v_i. By Theorem 1, it holds that even if the opposite edge $(v_{k+1}, v_k) \in E_i$, this is not sufficient to ensure the presence of (v_k, v_{k+1}) in G_i. Therefore, v_i will not include the latter in its graph construction. This contradicts the fact that $(v_k, v_{k+1}) \in E_i$.

(Sufficiency) The other direction is trivial, since if there exists $\overrightarrow{p} \in \mathcal{R}_i$, such that $(v_k, v_{k+1}) \subseteq \overrightarrow{p}$, then, (v_k, v_{k+1}) will be present in the graph constructed by v_i. □

An interesting observation is that the differences in the topological views of domains (such as the ones shown in Figure 19) might theoretically lead to inconsistencies like the one illustrated in Figure 21. In this example, c_i is a customer of P_i, $i = 1, \ldots, 3$, and provider P_2 is a peer of both P_1 and P_3. The effects of the VFR policies are shown in Figure 21b. Due to filtering policy FP2, P_2 will not provide transit between its peers. As a result, both P_3 and c_3 are unreachable from c_1. For symmetry, P_1 and c_1 are unreachable from c_3. Thus, from the perspectives of c_1 and c_3 the graph is disconnected. However, every vertex in the graph is reachable from c_2, so from the perspective of c_2, the graph is connected.

In what follows, we prove that the discrepancies in the topological views of domains are such that the graphs inferred by domains are in fact all mutually different, so the graph seen by each domain is unique.

Theorem 3. *In a routing system $\mathcal{S}$ with $|V| > 2$ domains subject to valley-free and local traffic distribution policies, where the routes are distributed between domains in the form of path vectors, the graphs that can be inferred by domains are all mutually different.*

Proof. Let v_i and v_j be two domains, such that $i \neq j$, $i, j \in \{1, \ldots, |V|\}$. There are two cases: 1) when v_j is not present in the path vectors known by v_i, or 2) when v_j is present in the path vectors known by v_i.

Consider Case 1, when v_j is not present in the path vectors known by v_i. This means that v_j is not reachable from v_i (one such case is illustrated in Figure 21 between c_1 and c_3). Thus, $v_j \nsubseteq G_i$, and since $v_j \subseteq G_j$, $G_i \neq G_j$.

Consider Case 2, when v_j is present in the path vectors known by v_i. This means that there exists at least one path vector $\overrightarrow{p}$ in $\mathcal{R}_i$, such that $\overrightarrow{p} = [\ldots, v_{j-1}, v_j, \ldots]$. Since $\overrightarrow{p}$ is in $\mathcal{R}_i$, then $\overrightarrow{p} \subset G_i$. On the contrary, the path vector $\overrightarrow{p}$ cannot be in $\mathcal{R}_j$, since according to the import rule (1) (see Section A), v_j will not import routes where v_j is contained in the path (this the usual way of avoiding forwarding loops in path vectors). Indeed, the directed edge $(v_{j-1}, v_j) \nsubseteq \overrightarrow{p_j}, \forall\, \overrightarrow{p_j}$ in $\mathcal{R}_j$. Since the directed edge (v_{j-1}, v_j) is not in $\mathcal{R}_j$, by Corollary 2 it holds that $(v_{j-1}, v_j) \nsubseteq G_j$. Clearly, the directed edge $(v_{j-1}, v_j) \subset \overrightarrow{p} \subset G_i$. Hence, $G_i \neq G_j$. This completes the proof. □

Corollary 4. *(Uniqueness of the graphs) Let $\mathcal{S}$ be a routing system with $|V| > 2$ domains subject to valley-free and local traffic distribution policies, where each domain v_i in $\mathcal{S}$, $i = 1, \ldots, |V|$, distributes routes in the form of path vectors. Let $\mathcal{G} =$*

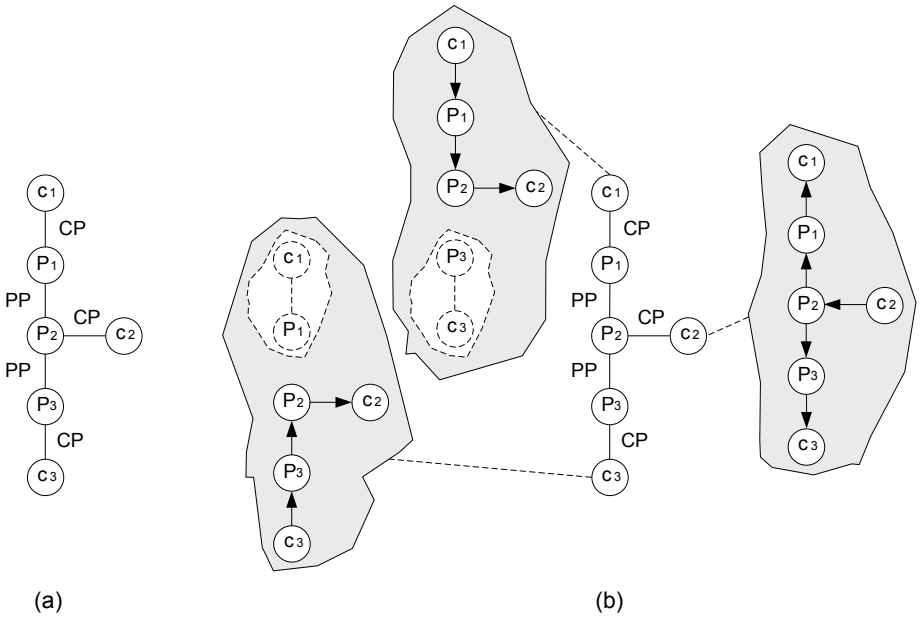


Figure 21: (a) AS graph G_0 with the commercial relationships between domains; (b) the graphs inferred by domains differ, leading to inconsistent views of the underlying graph G_0, to the extreme that, whilst every vertex is reachable from c_2, from the perspectives of c_1 and c_3, the graph is disconnected.

$\{G_1, \ldots, G_{|V|}\}$, *denote the set of graphs inferred by domains* $v_1, \ldots, v_{|V|}$, *respectively. Each graph* $G_i \in \mathcal{G}$ *is unique in* $\mathcal{G}$.

Proof. It follows immediately from Theorem 3. □

The main conclusion of this section is that, in an internetwork with $|V|$ domains that advertise path vectors subject to VFR policies, there are in fact $|V|$ different views of the underlying AS graph G_0. In the next section, we will show that although each of these views is inferred in accordance to the routing policies, the discrepancies in the topological views could still lead to inconsistencies in the paths available on the graphs constructed by domains.

4. Paths on graphs under routing policies

Let us now analyze to what extent the different topological views affect the availability of paths on the graphs inferred by domains. To this end, let us consider the topological views of the customers AS1 and AS4 in the example of Figure 19. These views are illustrated in Figure 22, where S_1 and S_4 represent the subscription graphs of AS1, and AS4, respectively. The subscription graph, S_i, of a domain v_i is composed of one vertex, namely, v_i, and the set of directed edges that connect v_i to its neighbors. In Figure 22, the graph S_1 is composed of the vertex AS1, and two directed edges connecting AS1 to its providers, (1,2), and (1,5) (see the bottom of Figure 22). Likewise, S_4 is composed of the vertex AS4, and two directed edges, (4,3), and (4,5) (see the top of Figure 22). In a link state-like paradigm, the topological view of AS1 is obtained by composing the graph S_1 with the advertisements received from AS2 and AS5. As shown at the bottom of Figure 22, this topological view depends on whether the graph computed by AS1 is assembled using the information received from AS2 or the information received from AS5.

On the one hand, AS1 computes the graph G_{12}, by composing S_1 with G_{21}^{adv}, where the latter represents the graph that AS1 can infer from the advertisements received from AS2. This composition is denoted as $G_{12} = S_1 \oplus G_{21}^{adv}$. As shown in Figure 22, the path AS1 $\rightarrow$ AS2 $\rightarrow$ AS3 $\rightarrow$ AS4 is feasible in G_{12}, since according to AS2, the link (2,3) must be present in the link-state database. On the other hand, AS1 computes the graph G_{15} with the information received from AS5. In G_{15}, the path AS1 $\rightarrow$ AS2 $\rightarrow$ AS3 $\rightarrow$ AS4 is unfeasible, since according to AS5, the link (2,3) must not be present in the link-state database. The absence of link (2,3) in G_{15} is due to the enforcement of filtering policy FP2 in AS2, since AS2 will not advertise routing information learned from AS3 to AS5. The top of Figure 22 shows that the same kind of inconsistencies occur in AS4 with the edge (5,2).

In summary, when it comes to the state of the links, the routing information received by AS1 and AS4 is inconsistent. Therefore, the topology and paths admitted by the routing policies cannot be captured and maintained using a standard link-state database.[9] Despite the inconsistencies, both AS1 and AS4 can still formally construct a graph of the network [21]. For instance, it can be easily checked that, from the valley-free routes contained in the first row of the NR table, $\mathcal{R}_1$ (see Table 5), AS1 can infer

[9] One of the main requirements in a link state-like paradigm is to warrant the consistency of the link state databases maintained by the nodes (domains) in the network. This is essential not only for constructing a consistent graph of the network, but also for finding paths on the graph.

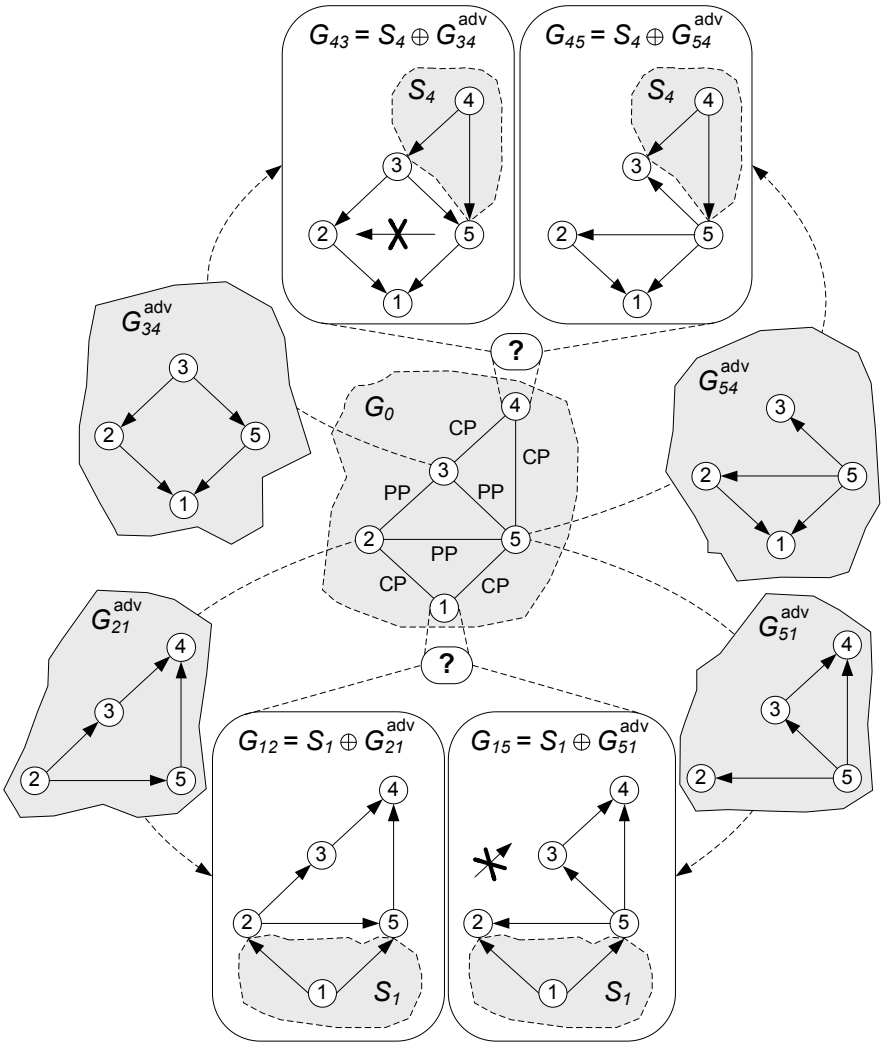


Figure 22: The VFR policies lead to inconsistencies in the topological views inferred by the customers AS1 and AS4.

a set of 8 adjacencies between domains and construct the graph G_1 shown in Figure 23.[10]

The interesting observation is that, although G_1 was inferred from $\mathcal{R}_1$, i.e., from the routes admitted by the routing policies, once constructed, the forwarding paths available on G_1 show inconsistencies with the paths allowed by the routing policies. We now formalize this observation.

Theorem 5. *Let v_i be a domain in a routing system $\mathcal{S}$ with $|V| > 2$ domains, and $\mathcal{R}_i$ be the corresponding row of v_i in table $\mathcal{R}$. Let G_i be the graph constructed by v_i from the path vectors contained in $\mathcal{R}_i$. For a path $\overrightarrow{p}$ to be in G_i, it is sufficient (but not necessary) that $\overrightarrow{p} \in \mathcal{R}_i$.*

[10]From Figures 22 and 23, it can be seen that the graph inferred by AS1 is in fact $G_{12} \cup G_{15}$—notice that we refer to the union, and not the "disjoint union" often used in graph theory [21].

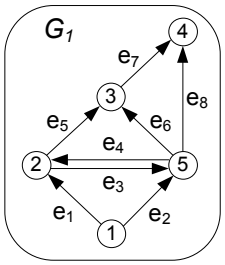


Figure 23: The graph inferred by AS1 from the valley-free routes contained in the first row of the NR table, $\mathcal{R}_1$ (cf. Table 5).

Proof. The sufficiency is trivial, since if $\overrightarrow{p} \in \mathcal{R}_i$, then by Corollary 2 it holds that, $\forall (v_k, v_{k+1}) \subseteq \overrightarrow{p} \Rightarrow (v_k, v_{k+1}) \in G_i$. Therefore, $\overrightarrow{p} \subset G_i$. To prove that it is not necessary, consider the example shown in Figure 23. Observe that G_1 admits the paths: [1,2,5,3], [1,5,2,3], [1,2,5,3,4], and [1,5,2,3,4], none of which is present in $\mathcal{R}_1$. Hence, there are paths in G_1 that are not present in $\mathcal{R}_1$. This completes the proof. □

The main conclusion from Theorem 5 is summarized in Table 6. So far we have shown that it is not possible to distributedly infer a "*single*", and "*consistent*", link state-like graph of a network from the VFR information known by domains. Another option is that domains obtain the interconnectivity graph, G_0, directly from a source dedicated to its computation. However, the graph G_0 per se is inadequate for routing purposes, since it suffers from the same inconsistencies shown in Table 6. For instance, from the graph G_0 used as example throughout this paper (cf. Figure 18a), it can be noted that many paths that are feasible in G_0, are actually not allowed by the routing policies; e.g., the path [1,2,5,3,4]. As a consequence, the conventional way of modeling an internetwork, as an AS graph with the vertex-set representing domains and the edge-set the interconnection among these latter, is inappropriate if the graph is to be used for routing purposes under VFR policies.

One alternative could be that domains obtain both G_0 and the AS relationships in G_0, $AS_r(G_0)$ (cf. Figure 18b). Leaving aside issues such as which administrations would have the competence to compute $\langle G_0, AS_r(G_0)\rangle$, and the authority to advertise it in the Internet, there are at least two arguments against this approach. First, Di Battista et al. [24] showed that, the problem of computing the types of relationships between domains from the analysis of routing tables is NP-complete. Thus, inferring $AS_r(G_0)$ from the routing information requires approximation algorithms, which

Table 6:

From the routing information known by a domain v_i	$\longrightarrow$	v_i can construct a link state-like graph G_i of the network
		$\downarrow$
G_i is not sufficient to reconstruct the paths known by v_i	$\longleftarrow$	However, once G_i is constructed

would entail a degree of inaccuracy in the tuple $\langle G_0, AS_r(G_0)\rangle$ advertised. In addition, Oliveira et al. [19] have recently shown the infeasibility to obtain a complete AS-level topology from current data collection efforts. It seems that to have an accurate picture of $\langle G_0, AS_r(G_0)\rangle$, domains would be required to make public their commercial relationships, which is precisely the second and most important argument against this approach. Indeed, many providers will probably not support the proposal of making public and maintaining updated the tuple $\langle G_0, AS_r(G_0)\rangle$.

In summary, what is missing is a mechanism through which domains can independently construct a graph, without disclosing $AS_r(G_0)$. Such graph must accurately abstract the effects of routing policies, and capture the AS-level connectivity and paths admitted by the VFR policies in a consistent way. A solution for these challenges is presented below.

5. Path-State Graphs (PSGs)

In this section, we introduce the concept of a Path-State Graph (PSG). As we shall show, the PSG is an instrument for compatibilizing the different topological views of domains, while keeping their routing policies as private as they are today. In this sense, the PSG bridges the theoretical gap between the link-state and policy-based routing paradigms (cf. challenge 1 in Section A).

Unlike link-state routing, a PSG keeps an AS-path level of detail of the network, since the link and router connectivity levels of detail should be irrelevant to the global routing system. For the sake of clarity, we will first provide the intuition behind the construction of the PSGs, guided through the same example used throughout the paper (cf. Figure 18b). After that, we will formally define the PSG and present its main properties.

To describe the construction of a PSG, let us consider again the example in Figure 18b. The PSG is inspired in the well-known concept of the *line graph* [21], so we start by illustrating the strengths and limitations of finding the line graph of the AS graph shown in Figure 18b. Figure 24a shows the AS graph G_0 (together with the AS relationships $AS_r(G_0)$), while Figure 24b shows its corresponding line graph $L(G_0) = (\bar{V}, \bar{E})$. The process for constructing the line graph is as follows. First, the links of the original graph G_0 are represented as nodes in $L(G_0)$. Second, two nodes of $L(G_0)$ must be adjacent if and only if their corresponding links are adjacent in graph G_0. For example, the nodes $\bar{v}_{23}$, $\bar{v}_{25}$, and $\bar{v}_{53} \in \bar{V}$ in the line graph in Figure 24b, represent the peering links (AS2, AS3), (AS2, AS5), and (AS5, AS3), respectively, in the AS graph in Figure 24a. Likewise, the edge $(\bar{v}_{25}, \bar{v}_{53})$ in Figure 24b, indicates that the

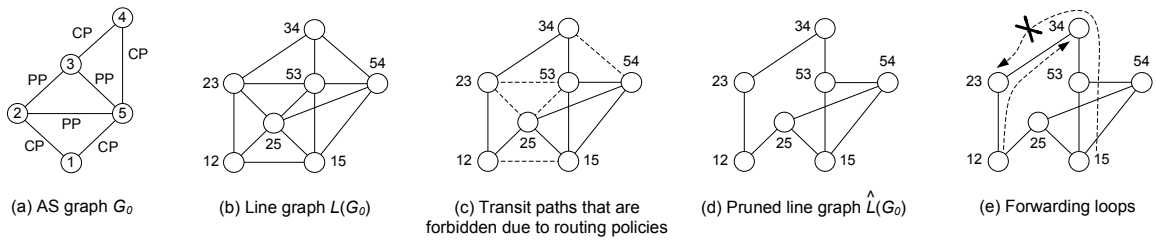


Figure 24: (a)–(d) Computing the line graph $L(G_0)$, and the pruned line graph, $\hat{L}(G_0)$, of the AS graph G_0; (e) the graph $\hat{L}(G_0)$ captures the transits that are allowed by the routing policies (summarized in Table 4), but some of the paths in $\hat{L}(G_0)$ might represent forwarding loops in G_0.

peering links (AS2, AS5) and (AS5, AS3) are adjacent in Figure 24a, since they are connected by AS5. The rest of nodes and edges in $L(G_0)$ follow the same construction principle.

It is important to observe that we have not yet applied any routing policy over the graphs shown in Figures 24a and 24b. The practical utility of the line graph is that, e.g., the edge $(\bar{v}_{25}, \bar{v}_{53})$ captures the potential transit between AS2 and AS3 through AS5, before the enforcement of the routing policies. More generally, the edges of the line graph capture the adjacencies—and therefore the possible transit—between the links of an AS graph. Accordingly, the dashed edges in Figure 24c represent the adjacencies that are forbidden in graph G_0 due to the filtering policies FP1 and FP2, whereas the solid ones represent the adjacencies that are allowed by FP3.

For example, the dashed edge $(\bar{v}_{25}, \bar{v}_{53})$ in Figure 24c indicates that the transit paths AS2 $\rightarrow$ AS5 $\rightarrow$ AS3 and its inverse AS3 $\rightarrow$ AS5 $\rightarrow$ AS2 in Figure 24a, will be filtered by AS5, since they are both forbidden due to the enforcement of filtering policy FP2 in AS5. The same occurs with the dashed edges $(\bar{v}_{23}, \bar{v}_{25})$ and $(\bar{v}_{23}, \bar{v}_{53})$ in Figure 24c, since they will be filtered by AS2, and AS3, respectively. In addition, the dashed edge $(\bar{v}_{12}, \bar{v}_{15})$ in Figure 24c indicates that the transit paths AS2 $\rightarrow$ AS1 $\rightarrow$ AS5 and its inverse AS5 $\rightarrow$ AS1 $\rightarrow$ AS2 in Figure 24a, will be filtered by AS1, since they are both forbidden due to the enforcement of filtering policy FP1 in AS1. The same occurs with the dashed edge $(\bar{v}_{34}, \bar{v}_{54})$, since AS4 will not provide transit between its providers.

By removing the forbidden adjacencies, we obtain the pruned line graph $\hat{L}(G_0)$ shown in Figure 24d. The latter has a remarkable strength: it only contains adjacencies permitted by the VFR policies. However, this graph representation has two problems that prevent its practical application.

On the one hand, $\hat{L}(G_0)$ is not opaque.[11] By this we mean that from $\hat{L}(G_0)$, AS5 can infer that AS2 and AS3 are interconnected—due to the presence of node $\bar{v}_{23}$ in $\hat{L}(G_0)$—which is generally against the interests of both AS2 and AS3. As shown in Figure 19, AS2 and AS3 will typically hide their mutual interconnection from AS5—recall that the topological view of AS5 is depicted as G_5 in Figure 19. Likewise, AS2 can infer from $\hat{L}(G_0)$ that AS5 and AS3 are connected, and AS3 would do the same for the connection between AS2 and AS5. All these crossed inferences are against the commercial interests of providers. The problem is that the AS relationships $AS_r(G_0)$ are inherently embedded in $\hat{L}(G_0)$, so they can be inferred from the latter.

On the other hand, the paths cannot be properly inferred from $\hat{L}(G_0)$, since the paths admitted in $\hat{L}(G_0)$ might represent forwarding loops in the AS graph G_0. Figure 24e shows one such example. While the path $\bar{v}_{12} \rightarrow \bar{v}_{23} \rightarrow \bar{v}_{34}$ in $\hat{L}(G_0)$ is feasible in G_0 (it corresponds to the AS-path AS1 $\rightarrow$ AS2 $\rightarrow$ AS3 $\rightarrow$ AS4 in G_0), the path $\bar{v}_{15} \rightarrow \bar{v}_{53} \rightarrow \bar{v}_{34} \rightarrow \bar{v}_{23}$ represents a loop in G_0 (AS1 $\rightarrow$ AS5 $\rightarrow$ AS3 $\rightarrow$ AS4 $\rightarrow$ AS3 $\rightarrow$ AS2). In summary, the AS-paths admitted in an AS graph cannot be inferred directly from its pruned line graph, and thus a loop avoidance mechanism is required.

On this basis, we seek to exploit the fact that a pruned line graph can effectively capture the link adjacencies permitted by the routing policies, and, at the same time, solve the two problems outlined before, i.e., ensure the opaqueness of the graphs, and eliminate the possibility of forwarding loops.

To achieve this, we propose the graph representations shown in Figure 25. In order to simplify the exposition, we only show the path-state graphs seen by AS2, and AS5, i.e., PSG_2, and PSG_5, respectively, as well as the path-state graph constructed by AS1,

[11] Later, in Section A we will formally define the concept of "opaqueness," and prove that the PSGs are opaque.

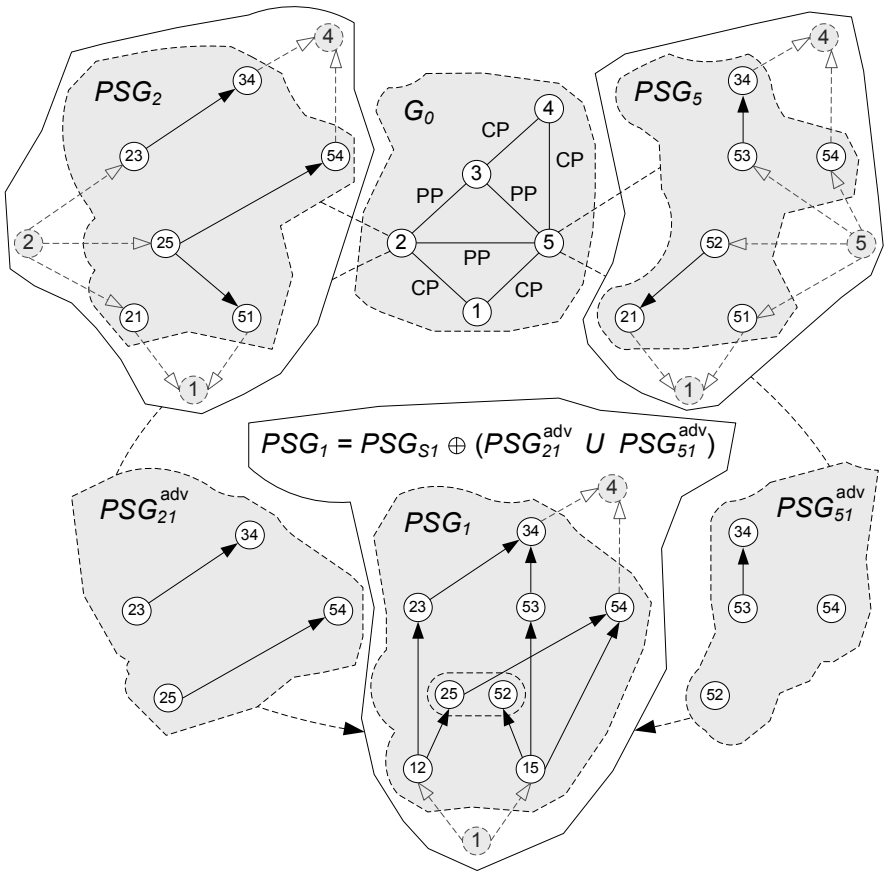


Figure 25: Construction of a PSG. The figure shows the different, although consistent, views of the AS-level relationships and paths admitted by the routing policies. Conversely to the case shown in Figure 22, now AS1's view can be consistently captured in a PSG, which is the composition of PSG_{S_1} with the union of the PSGs advertised by AS2 and AS5.

PSG_1. The key to obtain a compatible set of opaque graphs is that each domain independently computes a line graph, which can be locally shaped (e.g., pruned) before exporting it to a neighboring domain. Such graph shaping can be performed on a per neighbor basis, following the usual valley-free export policies of routing domains.

To fix ideas, the graphs PSG_2, and PSG_5, are the directed line graphs of the digraphs G_2, and G_5, in Figure 19, respectively; that is, $PSG_2 = L(G_2)$, and $PSG_5 = L(G_5)$. Furthermore, by observing Figures 22 and 25 together, it can be noted that the PSGs advertised to AS1 are $PSG_{21}^{adv} = L(G_{21}^{adv})$, and $PSG_{51}^{adv} = L(G_{51}^{adv})$. In addition, let PSG_{S_1} denote the subscription PSG of AS1, which is the line graph of S_1 (see Figure 22), i.e., $PSG_{S_1} = L(S_1)$. A subscription PSG, PSG_{S_i}, is a line graph composed solely by vertices, each of which represents a directed edge in graph S_i connecting domain v_i to one of its neighbors. For instance, $PSG_{S_1} = \{\bar{v}_{12}, \bar{v}_{15}\}$.

As shown in Figure 25, AS1 constructs its PSG by composing PSG_{S_1} with the PSGs received from its providers as follows: $PSG_1 = PSG_{S_1} \oplus (PSG_{21}^{adv} \bigcup PSG_{51}^{adv})$. The composition ($\oplus$) corresponds to the addition of a set of directed edges in PSG_1 of the form $(\bar{v}_{1x}, \bar{v}_{x-})$, where x denotes a neighbor of AS1. These edges simply connect the vertices $\bar{v}_{1x}$ introduced by PSG_{S_1}, with the set of vertices $\bar{v}_{x-}$ contained in PSG_{x1}^{adv}.

For example, in Figure 25, this composition introduces in PSG_1 the edges $(\bar{v}_{12}, \bar{v}_{23})$, $(\bar{v}_{12}, \bar{v}_{25})$, $(\bar{v}_{15}, \bar{v}_{52})$, $(\bar{v}_{15}, \bar{v}_{53})$, and $(\bar{v}_{15}, \bar{v}_{54})$.

To illustrate how the PSGs can be used for routing purposes, we have depicted (using dashed arrows in PSG_2 and PSG_5 in Figure 25) how AS2 and AS5 can reach the customers AS1 and AS4. We also show how AS1 can reach AS4 in PSG_1.

It is important to observe that the topological information exported to a neighboring domain will vary depending on the commercial relationship with that domain. For example, Figure 25 shows that AS2 and AS5 will export the graphs PSG_{21}^{adv}, and PSG_{51}^{adv}, respectively, to AS1. However, the PSGs exported directly between them will differ from these latter, since they will mutually hide their interconnection to AS3. Indeed, the topological information exchanged between each other will be pruned according to the valley-free export policies, in the form of the pruned line graphs $PSG_{25}^{adv} = L(G_{25}^{adv}) = \hat{L}_{25}(G_2)$, and $PSG_{52}^{adv} = L(G_{52}^{adv}) = \hat{L}_{52}(G_5)$. This information, together with the PSGs received from the rest of their corresponding neighbors, is what enables the independent construction of PSG_2 and PSG_5 at AS2, and AS5, respectively.

In addition, a straightforward way to avoid forwarding loops in the inference of AS-paths is by composing directed line graphs in the PSG constructions—in Section A we will formalize this property and prove that the PSGs are free from forwarding loops. For instance, PSG_1 solves the loop shown in Figure 24e, since in PSG_1, $\bar{v}_{15} \rightarrow \bar{v}_{53} \rightarrow \bar{v}_{34}$ is feasible, but $\bar{v}_{15} \rightarrow \bar{v}_{53} \rightarrow \bar{v}_{34} \rightarrow \bar{v}_{23}$ is not, given that once the vertex $\bar{v}_{34}$ is reached, there is no egress edge from it. This means that $\bar{v}_{34}$ is a terminal vertex, which is consistent with the fact that there may be no further transit for traffic going from AS3 to AS4.

One of the most important aspects of the PSGs shown in Figure 25, is that they can be locally constructed with information resulting from the enforcement of the usual VFR policies between domains. Another important aspect that is implicit in Figure 25, is that the PSGs constructed by domains differ between each other. This is essential to avoid disclosing $AS_r(G_0)$. The key is to leverage the advertisement of these different topological views of domains, which, as we will show, once assembled in a PSG, can consistently capture the valley-free paths while remaining opaque and loop-free.

We now proceed to formalize the intuition described above. We will first define the operations of union and composition of directed line graphs, and based on these define the PSG. After that, we will infer an adequate data structure for the PSGs, and prove that these latter solve the inconsistencies summarized in Table 6. The opaqueness and loop-free properties are proved in Section A, where we also provide a complexity bound for the data volume associated with the PSGs.

Definition 5. *(Union of Line Digraphs) The union of two directed line graphs (or line digraphs) $L' = (\bar{V}', \bar{E}')$ and $L'' = (\bar{V}'', \bar{E}'')$, is the line digraph $L = L' \bigcup L''$, whose vertex-set is $\bar{V} = \bar{V}' \bigcup \bar{V}''$ and the edge-set is $\bar{E}$, where a directed edge $(\bar{v}_{ij}, \bar{v}_{jk}) \in \bar{E}$ if and only if the single hop path $(\bar{v}_{ij}, \bar{v}_{jk}) \subset L' \vee (\bar{v}_{ij}, \bar{v}_{jk}) \subset L''$.*

Definition 6. *(Composition of Line Digraphs) The composition of two line digraphs, $L = (\bar{V}, \varnothing)$, whose edge-set is empty, and $L' = (\bar{V}', \bar{E}')$, is the line digraph $L \oplus L'$, whose vertex-set is $\bar{V} \bigcup \bar{V}'$ and the edge-set is $\bar{E} = \bar{E}' \bigcup \{(\bar{v}_{-x}, \bar{v}'_{x-})\}$, where $\bar{v}_{-x} \in \bar{V} \wedge \bar{v}'_{x-} \in \bar{V}'$.*

Definition 7. *(Path-State Graph) Let v_k and v_i be two neighboring domains in a routing system $\mathcal{S}$, and $L(S_i)$ be the line digraph of the subscription graph of v_i, S_i. Let G_k be the digraph that can be constructed by domain v_k from the path vectors contained in $\mathcal{R}_k$. Let $\hat{L}_{ki}(G_k)$ be the line digraph advertised from v_k to v_i, pruned by v_k according to the VFR policies in $\mathcal{S}$. The Path-State Graph PSG_i constructed by domain v_i is the graph resulting from the composition $PSG_i = L(S_i) \oplus \bigcup_k \hat{L}_{ki}(G_k)$.*

Table 7: The Path-State Table (PST) $\mathcal{P}$.

	1	2	3	4	5
1	-	(12) [(15),(52)]	[(12),(23)] [(15),(53)]	[(12),(23),(34)] [(15),(54)] [(12),(25),(54)] [(15),(53),(34)]	(15) [(12),(25)]
2	(21) [(25),(51)]	-	(23)	[(23),(34)] [(25),(54)]	(25)
3	[(32),(21)] [(35),(51)]	(32)	-	(34) [(35),(54)]	(35)
4	[(43),(32),(21)] [(45),(51)] [(43),(35),(51)] [(45),(52),(21)]	[(43),(32)] [(45),(52)]	(43) [(45),(53)]	-	(45) [(43),(35)]
5	(51) [(52),(21)]	(52)	(53)	(54) [(53),(34)]	-

5.1. Storing and Maintaining the PSGs

Providing an adequate data structure for storing and maintaining the PSGs is straightforward. Consider again the network routing table $\mathcal{R}$ shown in Table 5. Only two operations are needed to transform the path vectors contained in Table 5 into the routing records shown in Table 7—which henceforth shall be referred to as the *Path-State Table* (PST) $\mathcal{P}$. These two operations are basically the following. Consider a path vector $\overrightarrow{p} = [v_{i+1}, v_{i+2}, v_{i+3} \ldots, v_{k-1}, v_k] \subset \mathcal{R}_i$, where $\mathcal{R}_i$ denotes the corresponding row of domain v_i in Table 5. By inserting v_i as the first element of $\overrightarrow{p}$, $\forall\ \overrightarrow{p} \in \mathcal{R}_i$ (first operation), and then splitting each path vector into a vector of adjacencies between edges, i.e., $\overrightarrow{p_l} = [(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), (v_{i+2}, v_{i+3}), \ldots,$
$(v_{k-1}, v_k)] = [\bar{v}_{i(i+1)}, \bar{v}_{(i+1)(i+2)}, \bar{v}_{(i+2)(i+3)}, \ldots, \bar{v}_{(k-1)k}]$ (second operation), the routing records in table $\mathcal{R}$ adopt the form of the ones shown in table $\mathcal{P}$. These simple operations transform the routing information that a domain v_i can use to construct a link state-like graph G_i, into information that can be used to construct a path-state graph PSG_i. For instance, notice that the paths contained in rows 1, 2, and 5 of Table 7, match the paths captured by PSG_1, PSG_2, and PSG_5, respectively, in Figure 25. The transformation is denoted as $\mathcal{P} = \mathcal{T}(\mathcal{R})$, and the inverse operations as $\mathcal{R} = \mathcal{T}^{-1}(\mathcal{P})$.

It is worth noticing that the two operations for transforming table $\mathcal{R}$ into table $\mathcal{P}$ are essentially: 1) the insertion of domain v_i and linking it with its neighbors v_{i+1}, i.e., the composition ($\oplus$) in Definition 7; and 2) converting paths on digraphs into paths in line digraphs—we omit the details, but from the construction of the global NR table $\mathcal{R}$, it can be shown that $\bigcup_k \hat{L}_{ki}(G_k) = \hat{L}(\bigcup_k G_{ki}^{adv})$.

We now present one of the main contributions of this paper, which is the proof that the PSGs consistently capture the paths available in an internetwork subject to VFR policies.

Theorem 6. *(Paths consistency) Let v_i be a domain in a routing system $\mathcal{S} = \langle G_0, \mathcal{R} \rangle$, where the network routing table $\mathcal{R}$ contains the set of routing records resulting from the application of the VFR policies on G_0. Let $\mathcal{P} = \mathcal{T}(\mathcal{R})$ denote the transformation of the path vectors in table $\mathcal{R}$ into the routing records in table $\mathcal{P}$. Let $\mathcal{R}_i = \mathcal{T}^{-1}(\mathcal{P}_i)$ be the corresponding row of v_i in table $\mathcal{R}$, and PSG_i be the path-state graph constructed by v_i from the paths contained in row $\mathcal{P}_i$ of table $\mathcal{P}$. A path $\overrightarrow{p_l}$ is contained in PSG_i if and only if its corresponding path vector $\overrightarrow{p} = \mathcal{T}^{-1}(\overrightarrow{p_l}) \subset \mathcal{R}_i$.*

Proof. (Sufficiency) For each path $\overrightarrow{p} \subset \mathcal{R}_i$, it holds that there is a path $\overrightarrow{p_l}$ such that $\overrightarrow{p_l} = \mathcal{T}(\overrightarrow{p}) \subset \mathcal{P}_i$. Since PSG_i is constructed by v_i from the paths contained in $\mathcal{P}_i$, $\overrightarrow{p_l} \subset PSG_i$. Hence, if $\overrightarrow{p} \subset \mathcal{R}_i \Rightarrow \overrightarrow{p_l} \subset PSG_i$.

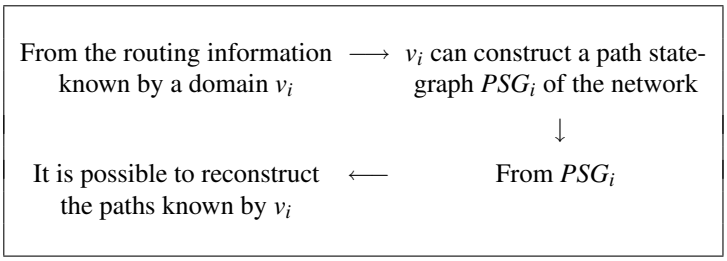
(Necessity) Suppose toward a contradiction that there exists a path $\overrightarrow{p_l} \subset PSG_i$, such that $\overrightarrow{p} = \mathcal{T}^{-1}(\overrightarrow{p_l}) \nsubseteq \mathcal{R}_i$. By construction, table $\mathcal{R} = \mathcal{V}(G_0)$, so $\mathcal{R}_i$ contains all the valley-free routes known by domain v_i. Since $\overrightarrow{p} \nsubseteq \mathcal{R}_i \Rightarrow \overrightarrow{p}$ is not valley-free, so $\overrightarrow{p_l}$ is also not valley-free. This would mean that there exists a non valley-free path $\overrightarrow{p_l}$ in PSG_i. Hence, there must be at least one edge $(\bar{v}_{xy}, \bar{v}_{yz})$ in $\overrightarrow{p_l} \subset PSG_i$, such that the transit AS$x \rightarrow$ AS$y \rightarrow$ ASz is not allowed in G_0 due to the VFR policies. From Definition 7, there are two possibilities: 1) $(\bar{v}_{xy}, \bar{v}_{yz}) \subset \bigcup_k \hat{L}_{ki}(G_k)$, or 2) $(\bar{v}_{xy}, \bar{v}_{yz}) \nsubseteq \bigcup_k \hat{L}_{ki}(G_k)$ but arises from the composition operation ($\oplus$).

Consider the first possibility, i.e., $(\bar{v}_{xy}, \bar{v}_{yz}) \subset \bigcup_k \hat{L}_{ki}(G_k)$. From Definition 5, it holds that the edge $(\bar{v}_{xy}, \bar{v}_{yz})$ can only be in the union if the single hop path $(\bar{v}_{xy}, \bar{v}_{yz})$ is contained in at least one line digraph $\hat{L}_{ki}(G_k)$. Thus, at least one of the line digraphs advertised to domain v_i contains a non valley-free edge. By induction to the origin, it is straightforward to show that this contradicts Definition 7, since the line digraphs are pruned according to the VFR policies before being advertised to a neighboring domain.

Assume now that $(\bar{v}_{xy}, \bar{v}_{yz}) \nsubseteq \bigcup_k \hat{L}_{ki}(G_k)$, and consider the second possibility, i.e., $(\bar{v}_{xy}, \bar{v}_{yz})$ arises from the composition operation ($\oplus$). This means that, the insertion of the vertices $\bar{v}_{iy}$ in the subscription PSG of domain v_i generates at least one edge $(\bar{v}_{iy}, \bar{v}_{yz})$ that is not valley-free. Hence, the transit AS$i \rightarrow$ AS$y \rightarrow$ ASz is not allowed in G_0 due to the filtering policies FP1–FP3 in ASy. Once again, this contradicts Definition 7, since ASy will not advertise to ASi a path toward ASz. This completes the proof. □

The most important contribution in this section is that the PSG solves the inconsistencies exemplified in Figure 22 and summarized Table 6. Whereas the topological views of domains cannot be abstracted as a single link state-like graph under VFR policies, they can be captured through a set of PSGs, from which the AS-paths can be consistently inferred (cf. Theorem 6). This is summarized in Table 8.

Table 8:



6. Main Properties of the Path-state Graphs

In this section, we formalize the concept of opaqueness in terms of routing policies, and prove that the PSGs are both opaque and loop-free. We also analyze here the state complexity of the PSGs in the context of scale-free graphs.

Definition 8. *(Opaqueness of a graph) Let v_i be a domain in a routing system $\mathcal{S}$, and $\mathcal{R}_i$ be the set of valley-free paths known by v_i through $\mathcal{S}$. Let G_i be the graph representation of the internetwork constructed by v_i. The graph G_i is said to be opaque if every path available on G_i is contained in $\mathcal{R}_i$.*

In essence, the condition that a graph G_i must satisfy to be opaque is that, the paths on G_i must be restricted to the valley-free routes distributed by $\mathcal{S}$ that reach v_i. This means that the AS relationships that can be inferred by v_i are strictly those embedded in the valley-free routes received at v_i.

Property 7. *The PSGs are opaque.*

Proof. Let PSG_i be the path-state graph constructed by v_i. By Theorem 6, it holds that a path $\overrightarrow{p_l} \subset PSG_i \Leftrightarrow \overrightarrow{p} = \mathcal{T}^{-1}(\overrightarrow{p_l}) \subset \mathcal{R}_i$, so every path in PSG_i is in $\mathcal{R}_i$. Hence, PSG_i is opaque. □

Property 8. *The PSGs are loop-free.*

Proof. Let PSG_i be the path-state graph constructed by v_i of an AS graph G_0. Suppose toward a contradiction that there exists a path $\overrightarrow{p_l} \subset PSG_i$, such that $\overrightarrow{p} = \mathcal{T}^{-1}(\overrightarrow{p_l})$ represents a loop in G_0. By Theorem 6, it holds that the path $\overrightarrow{p}$ must be contained in $\mathcal{R}_i$. However, $\overrightarrow{p} \nsubseteq \mathcal{R}_i$, since by construction (see the import rule (1) in Section A), the NR table $\mathcal{R}$ is free from routing loops—the domains in G_0 will never import routes containing loops into table $\mathcal{R}$. □

6.1. State Complexity

Before obtaining a complexity bound for the PSG, it is important to remark the philosophy behind its construction. Each domain is represented as a single entity, where the router and link levels of detail are omitted. If two neighboring domains are connected through multiple links, they are all abstracted as a single connection. Indeed, the vertices of a PSG, $\bar{v}_{ij}$, can be thought as the aggregate of the eBGP sessions between two neighbors ASi and ASj, whilst the edges, $(\bar{v}_{ij}, \bar{v}_{jk})$, aggregate the iBGP sessions within ASj for the transit AS$i \rightarrow$ AS$j \rightarrow$ ASk. Thus, the PSG offers a rather stable topological structure that will vary upon changes such as, a new domain is connected, a domain is removed, all the connections between two neighboring domains fail, two non-adjacent domains become peers, etc.

In addition, we observe that for many scale-free graphs of interest in practice, the node degree exponent, β, is in the range $\beta \leq -2$. For example, the value of β for the Internet's AS graph has stayed at approximately $\beta \approx -2.2$ since 1998 [25, 26, 27]. Hence, we shall focus on scale-free graphs with an exponent β in this range. Under these assumptions, the state complexity of the PSG can be upper bounded as follows.

Theorem 9. *Let $\mathcal{S} = \langle G_0, \mathcal{R} \rangle$ be a routing system, such that $G_0(V,E)$ is a scale-free graph with node degree exponent $\beta \leq -2$. Let k be the number of different frequencies of node degrees in G_0, and PSG_i be the path-state graph constructed by a node $v_i \in V$. The state complexity of the path-state graph PSG_i is $O\left(|V|k^{\beta+3}\right)$.*

The proof of Theorem 9 can be found in the Appendix.

Remark 1. *For many scale-free graphs, the node degree distribution is sparse, often with $k \ll |V|$. For example, while the current number of nodes in the Internet's AS graph is $|V| \approx 34.000$, there are only $k = 198$ different frequencies of node degrees in this graph.*

To illustrate what this upper bound represents in practice, we have examined the data volume that would need to be managed to maintain the PSG of an Internet-size scale-free graph. Indeed, we have examined which would have been the growth of such PSG from 1998 until today. Figure 26 shows the evolution of the state complexity of the PSG for different values of the exponent β. The values of β were extracted from relevant works in the area (see e.g., [25, 26, 27]), and the state complexity was approximated using inequality (14) (see the Appendix). Figure 26 also shows the evolution of the sizes of the BGP Routing Information Base (RIB), and the BGP Forwarding Information Base (FIB) in the Default-Free Zone (DFZ) since 1998. The important observation is that the theoretical growth of the size in the state of an Internet-size PSG would have remained consistently between the BGP FIB and RIB sizes from 2000 until today.

Remark 2. *Figure 26 shows that the growth of the upper bound complexity is slightly super-linear. Conversely to the dynamism present in a BGP RIB, the PSG offers a rather stable interconnection graph, which will change in practice in much slower timescales than the data in the BGP RIB. In Part II of this paper, we develop an interdomain protocol suite, called PSP/BGP (Path-State Protocol/Border Gateway Pro-*

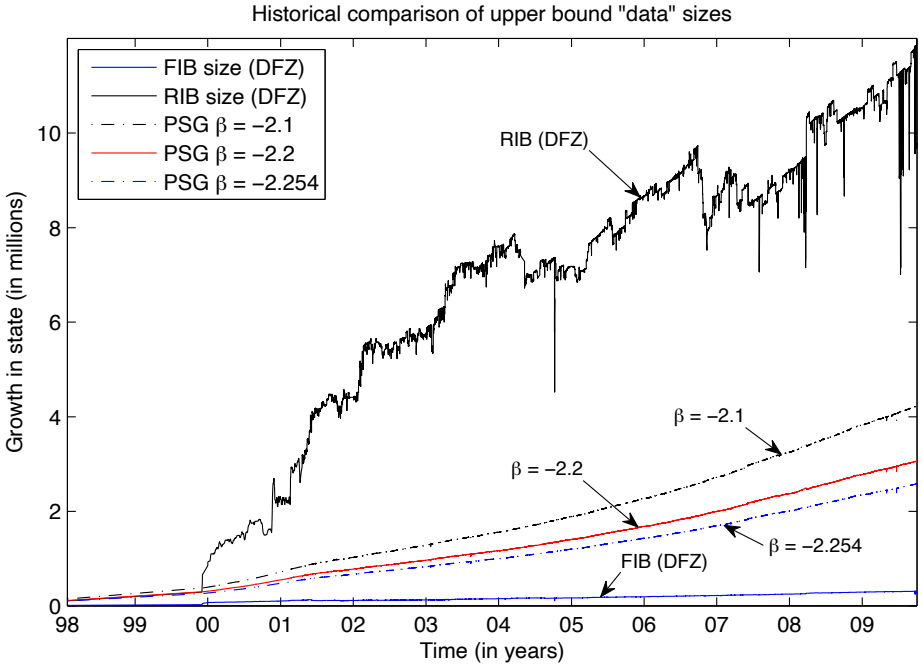


Figure 26: Evolution of the data volume associated with the routing state of the BGP FIB and RIB, and the PSG for different node degree exponents β.

tocol), to exploit the PSGs. Based on compact routing theory [8], we shall prove that the complexity bound for the dynamics in this protocol suite is optimal on scale-free graphs, and show that this is possible when a set of Internet domains maintain the state shown in Figure 26.

7. Related Work

Different initiatives have advocated in favor of combining the strengths of link-state and path vectors in a unified routing model. The closest related work to ours is that of White [1], which proposes to develop graph overlays on path vectors. The author discusses some of the potential benefits of overlaying an AS-level graph over the BGP protocol, such as faster convergence upon path failures, improved security, and the possibility of conveying information about the interconnections and policies between domains. Our work is largely inspired in [1].

An interesting alternative can be found in [2], where Subramanian et al. present a hybrid link-state and path vector protocol called HLP. Despite its strengths, HLP does not solve the challenges exposed in Section A. Differently from our proposal, HLP is devised as a replacement to BGP (against the practical challenge). Moreover, HLP proposes to use a link state-like routing model only within a provider-customer hierarchy, and a fragmented path vector protocol across hierarchies. With this approach, each link-state hierarchy is only subject to policy FP3. As shown in Figure 22, the inconsistencies and challenges arise when link-state routing needs to deal with policies FP1 and FP2. As a result, HLP does not solve the theoretical challenge either, since the complexity in blending path vectors and link-state routing is to deal with policies FP1 and FP2.

With a different goal, Sprintson et al. [28] propose a graph representation of an internetwork supporting the optimal computation of two link-disjoint paths spanning multiple IP/MPLS domains, both in the absence and presence of routing policies. Our work builds upon some of the well-known tools used in that work, but differs from it both in scope and reach. The optimality of the routing mechanism provided in [28] entails a router-level of detail in the graph representation. This granularity comes at the cost of the complexity of the state that needs to be stored and distributed by each domain. Instead, the PSG introduced here operates at the AS level, and we have shown that even maintaining the PSG of an Internet-size scale-free graph is manageable in practice (cf. Figure 26). In addition, the graph representation proposed in [28] is not opaque. The path-state graph introduced here solves this problem.

8. Conclusion

During years, many research papers and books have modeled an internetwork as a single graph $G(V,E)$, with the set of vertices V representing domains and the edges E the interconnectivity of the latter. In this paper, we have firstly examined the applicability of these graphs in the context of path vector routing under routing policies. The results of our analysis can be summarized as follows.

In an internetwork with $|V|$ domains subject to valley-free routing policies, there are actually $|V|$ different views of the underlying graph $G(V,E)$, since the topological views of domains are all different from each other. As a result, the forwarding paths admitted by the routing policies can neither be captured through these views nor

through the underlying graph $G(V,E)$—at least not consistently. Therefore, conventional AS graph representations of an internetwork are unsuitable for routing purposes under valley-free routing policies.

To solve the inconsistencies exposed, we proposed a new graph representation called a Path-State Graph (PSG). Each domain can independently construct a PSG by collecting the different topological views advertised by its neighbors. These advertisements are sent according to the usual export policies of routing domains. The PSGs constructed by domains are in fact different from each other, which is a consequence of the need for keeping private the business relationships and peering policies between domains. One of the main advantages of these graphs is that they can combine different pieces of topological information, which, once assembled, accurately capture the valley-free paths available in the network. Another major advantage is that the process for constructing the PSG-set is entirely distributed, while preserving key properties, such as the assurance that the PSGs are both opaque and loop-free.

As discussed above, this paper has shown that, in the context of policy-based path vectors, formalisms such as: "*The network is modeled as a graph* $G(V,E)$*...*" or variations of it, lack of any practical utility when it comes to finding paths on conventional graphs $G(V,E)$. We argue that, for such cases, a more precise claim could be: "*The network is modeled as a set of path-state graphs...*".

Another important aspect is that, due to the hierarchical nature of the customer-provider relationships between domains, it is straightforward to prove that the PSGs are indeed directed acyclic graphs (DAGs).[12] This means that the properties of DAGs apply to the PSGs, e.g., the path-counting problem between two vertices in the PSG can be solved in linear time.

At this stage, we have focused on the construction of graphs at the AS-level. The possibility of increasing the granularity at the IP prefix level might bring new options, e.g., in terms of traffic engineering, though at the cost of increased complexity in the state of the PSGs. For instance, the local traffic distribution policies applied by a multi-homed domain u, may cause that the rest of domains (or a subset of these) end up inferring different PSGs for different prefixes originated at u.

Overall, the graphs introduced in this work offer a remarkably versatile tool for modeling, since a graph that concurrently: a) captures policies; b) provides different, although consistent, views of the adjacencies and relationships among a set of entities; c) without violating the opaqueness required by these entities, can be exploited in many different fields and contexts, including the management of different security levels of information, distributed management of business information, etc.

In Part II, we shall focus on the non-disruptive integration of the PSGs and the BGP protocol. In particular, we will provide PSG-based mechanisms to optimize the routing dynamics on scale-free graphs, and describe potential applications of the PSGs in terms of path-authentication, and traffic engineering.

[12]The acyclic nature of the graphs arises because if a domain u is a customer of provider v, which is in turn a customer of provider w, then the latter will not be a customer of u in practice.

Proof of Theorem 5

Proof. Let $PSG_i = (\bar{V}_i, \bar{E}_i)$ be the path-state graph of $G_0(V,E)$, constructed by a domain $v_i \in V$ from the valley-free routes known by the latter. The state complexity of PSG_i is $|\bar{V}_i| + |\bar{E}_i|$, hence we seek an upper bound for this sum.

Let d_j, $j = 1,\ldots,|V|$, denote the degree of node v_j in graph G_0. For each node $v_j \in V$, the maximum possible number of corresponding edges, $(\bar{v}_{-j}, \bar{v}_{j-})$, in PSG_i, is given by the permutations $P_2^{d_j} = \frac{d_j!}{(d_j-2)!}$. This is the case when the transit between every pair of edges adjacent to v_j in G_0 must be present in PSG_i. Clearly, the minimum possible number of corresponding edges for v_j in PSG_i is zero, which occurs, e.g., when v_j is a non-transit domain, or when v_j is not reachable from v_i (see Figure 21).

Let $X\colon\ E \times E \to \{0,1\}$, be a test function, such that for any pair of adjacent edges $e, e' \in E \subset G_0$, $X(e,e') = 1$ if the transit $e \to e'$ is of type $(\overrightarrow{\text{PP}} \to \overrightarrow{\text{PP}}) \vee (\overrightarrow{\text{PC}} \to \overrightarrow{\text{CP}}) \vee (\overrightarrow{\text{PP}} \to \overrightarrow{\text{CP}}) \vee (\overrightarrow{\text{PC}} \to \overrightarrow{\text{PP}})$, and $X(e,e') = 0$ otherwise. In other words, the test function X indicates whether the transit $e \to e'$ is not valley-free. The test X for the transit $(v_n, v_j) \to (v_j, v_m)$ is denoted as $X((v_n,v_j),(v_j,v_m)) = x_{nm}^{(j)}$. In addition, let $Y\colon V \times E \times E \to \{0,1\}$, be a binary function such that for any vertex $v \in V$ and any pair of adjacent edges $e, e' \in E \subset G_0$, $Y(v,e,e') = 1$ *iff* $X(e,e') = 0$ and the transit $e \to e'$ is not contained in the path vectors known by domain v. This means that, within the set of transits $e \to e'$ that are valley-free, the function Y tests whether they must be contained in PSG_v. The test Y for the transit $(v_n, v_j) \to (v_j, v_m)$ in PSG_i is denoted as $Y(v_i,(v_n,v_j),(v_j,v_m)) = y_{nm}^{(j,i)}$.

To fix ideas, consider AS5 and AS1 in Figure 25. For AS5 it holds that, e.g., $x_{23}^{(5)} = 1$, whilst $x_{43}^{(5)} = 0$. From the perspective of AS1, on the other hand, $y_{43}^{(5,1)} = 1$, since the transit AS4 $\to$ AS5 $\to$ AS3 is valley-free, but this transit path must not be contained in PSG_1 (notice that the path [4,5,3] is not contained in $\mathcal{R}_1$). Indeed, from the $\frac{4!}{2!} = 12$ possible transits through AS5, only 4 are present in PSG_1 (see Figure 25). The rest, either yield true in test X or in test Y, so they must not be present in PSG_1.

Let $V_T \subset V$, denote the set of transit domains in graph G_0. Accordingly, the number of directed edges $|\bar{E}_i|$ in PSG_i is:

$$|\bar{E}_i| = \sum_{j=1}^{|V_T|} \Big[\frac{d_j!}{(d_j-2)!} - \sum_{n=1}^{d_j} \sum_{\substack{m=1 \\ m \neq n}}^{d_j-1} x_{nm}^{(j)} + y_{nm}^{(j,i)} \Big] \tag{4}$$

which can be bounded as follows (see Remark 3 at the end):

$$|\bar{E}_i| < \sum_{j=1}^{|V_T|} d_j^2 \leq \sum_{j=1}^{|V|} d_j^2 \tag{5}$$

The number of nodes with degree d is given by the frequency f_d, and since $f_d \propto d^{\beta}$, the probability that a node has degree d is thus:

$$p(d) = \frac{f_d}{|V|} = \frac{\mathcal{C}}{|V|}\, d^{\beta} \tag{6}$$

where $\mathcal{C}$ is the proportionality constant in the degree power-law in graph G_0. By sorting in the order of increasing degree, and grouping terms in the k different frequencies of degrees, the summation on the right-hand-side of inequality (5) can expressed as:

$$S_1 = \sum_{j=1}^{|V|} d_j^2 = f_{d_1} d_1^2 + \cdots + f_{d_k} d_k^2 \tag{7}$$

$$\Rightarrow \quad \frac{S_1}{|V|} = p(d_1)d_1^2 + \cdots + p(d_k)d_k^2 = \sum_{l=1}^{k} p(d_l)d_l^2 \tag{8}$$

Hence, the bound in (5) can be expressed as follows:

$$|\bar{E}_i| < |V| \sum_{l=1}^{k} p(d_l)d_l^2 = \mathcal{C} \sum_{l=1}^{k} d_l^{\beta+2} \tag{9}$$

where the proportionality constant $\mathcal{C}$ can be derived by normalizing the probabilities in (6). Accordingly:

$$\mathcal{C} = \frac{|V|}{\sum_{l=1}^{k} d_l^{\beta}} \Rightarrow |\bar{E}_i| < |V| \frac{\sum_{l=1}^{k} d_l^{\beta+2}}{\sum_{l=1}^{k} d_l^{\beta}} \tag{10}$$

Due to the sorting and grouping of node degrees in the different frequencies in (7), it holds that $d_l \geq l$, $\forall\ l \in \{1 \ldots k\}$, so when $\beta \leq -2$, the powers of the degrees in the numerator of the right-hand side of the inequality (10), can be bounded by: $d_l^{\beta+2} \leq l^{\beta+2}$. On the other hand, the series in the denominator in (10) converges for $\beta \leq -2$, and given that the sequence of the series is positive, it holds that $\sum_{l=1}^{k} d_l^{\beta} \geq d_1^{\beta}$, $\forall\ k > 1$. In light of the above, (10) can be bounded as follows:

$$|\bar{E}_i| < d_1^{-\beta} |V| \sum_{l=1}^{k} l^{\beta+2} \tag{11}$$

Since in (7) the degrees are sorted in the order of increasing degree, without loss of generality we assume that for the first frequency, the degree is $d_1 = 1$. Thus,

$$|\bar{E}_i| < |V| \sum_{l=1}^{k} l^{\beta+2} \tag{12}$$

It is easy to observe that the sequence in the series in (12) is positive and non-increasing for $\beta \leq -2$, so the integral test for convergence can be applied. Let $r(x)$ be an integrand function of the form $r(x) = x^{\beta+2}$, $r[1, +\infty) \rightarrow [0, +\infty)$. Then, for each $k \in \mathbb{N}$, it holds that:

$$\sum_{l=1}^{k} r(l) = \int_1^k r(x)dx + C_r + \varepsilon_r(k) \tag{13}$$

with C_r constant, and $0 \leq \varepsilon_r(k) \leq r(k) - \lim_{x \rightarrow +\infty} r(x)$. Since $\beta \leq -2$, the limit of $r(x)$ when $x \rightarrow +\infty$ is zero when $\beta < -2$, and one if $\beta = -2$. Thus, when $\beta < -2$, $\varepsilon_r(k) \leq r(k) \Rightarrow \varepsilon_r(k) = O\left(k^{\beta+2}\right)$, and $\varepsilon_r(k) = 0$ if $\beta = -2$.

By integrating in (13) and using (12) we obtain:

$$|\bar{E}_i| < |V| \left(\frac{k^{\beta+3} - 1}{\beta + 3} + C_r + \varepsilon_r(k) \right) \tag{14}$$

Therefore, $|\bar{E}_i| = O(|V| k^{\beta+3})$. On the other hand, the number of vertices $|\bar{V}_i|$ in PSG_i is bounded by twice the number of links in G_0, which is given by:

$$|\bar{V}_i| = \sum_{j=1}^{|V|} d_j \tag{15}$$

Following the same line of argument as above, it holds that $|\bar{V}_i| = O(|V|k^{\beta+2})$. Therefore, the state complexity of PSG_i is $O(|\bar{V}_i| + |\bar{E}_i|) = O(|V|k^{\beta+3})$. This completes the proof. □

Remark 3. *The upper bound in (5) is very conservative. Finding a tighter bound is an open problem deserving further work, where the challenge is to quantify the negative terms in (4). For instance, the terms $y_{nm}^{(j,i)}$ involve a reachability problem in node v_i, which Griffin et al. [22] showed to be NP-complete.*

Acknowledgment

We would like to thank Russ White, Fred Baker, Rodolfo Milito, and Pere Monclus from Cisco Systems Inc., for their support and guidance during the development of this work. We would also like to thank Alex Sprintson for many valuable discussions.

References

[1] R. White. Graph Overlays on Path Vector: A Possible Next Step in BGP. *Internet Protocol Journal, Cisco*, 8(2):13–21, June 2005.

[2] L. Subramanian, M. Caesar, C. Tien Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A Next-generation Interdomain Routing Protocol. *ACM/SIGCOMM*, Philadelphia, PA, USA, August 2005.

[3] Quagga Routing Suite: http://www.quagga.net.

[4] D. Pei, D. Massey, and L. Zhang. Finite State Machines for BGP. Csd tr-040047, UCLA, CA, USA, November 2004.

[5] O. Bonaventure, C. Filsfils, and P. Francois. Achieving sub-50 milliseconds recovery upon BGP peering link failures. *IEEE/ACM Trans. on Networking*, 15(5):1123–1135, October 2007.

[6] R. White. Securing BGP Through Secure Origin BGP. *The Internet Protocol Journal*, 6:15–22, June 2003.

[7] M. Zhao, S. W. Smith, and D. M. Nicol. The Performance Impact of BGP Security. *IEEE Network*, 19(6):42–48, Nov./Dec. 2005.

[8] D. Krioukov, KC Claffy, K. Fall, and A. Brady. On compact routing for the internet. *ACM SIGCOMM Computer Communications Review*, 37(3):43–52, July 2007.

[9] A. Dhamdhere and C. Dovrolis. Ten Years in the Evolution of the Internet Ecosystem. In *Proc. of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC)*, Vouliagmeni, Greece, October 2008.

[10] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). IETF RFC 4271, January 2006.

[11] N. Feamster, H. Balakrishnan, and J. Rexford. Some Foundational Problems in Interdomain Routing. In *Proc. of ACM SIGCOMM HotNets-III*, San Diego, CA, USA, November 2004.

[12] C. Labovitz, A. Ahuja, A. Abose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Trans. on Networking*, 9(3):293–306, June 2001.

[13] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. IETF RFC 4984, September 2007.

[14] R. Oliveira, B. Zhang, D. Pei, R. Izhak-Ratzin, and L. Zhang. Quantifying Path Exploration in the Internet. *IEEE/ACM Trans. on Networking*, 17(2):445–458, April 2009.

[15] B. Quoitin. BGP-based Interdomain Traffic Engineering. *Doctoral Thesis, Louvain-la-Neuve, Belgium*, 2006.

[16] W. Aiello, J. Ioannidis, and P. McDaniel. Origin authentication in interdomain routing. In *Proc. of the 10th ACM conference on Computer and communication security*, Washington D.C., USA, October 2003.

[17] S. Murphy. BGP Security Vulnerabilities Analysis. IETF RFC 4272, January 2006.

[18] CAIDA's AS Relationships Dataset: http://www.caida.org/data/active/as-relationships/.

[19] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. The (in)Completeness of the Observed Internet AS-level Structure. *IEEE/ACM Trans. on Networking*, to appear in 2010.

[20] L. Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Trans. on Networking*, 9(6):733–745, December 2001.

[21] J. L. Gross and J. Yellen. Graph Theory and its Applications. 2nd edition, Champan & Hall/CRC, 2006.

[22] T. G. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. *ACM/SIGCOMM*, Cambridge MA, USA, August 1999.

[23] The Route Views Project: http://www.routeviews.org.

[24] G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and Thomas Schank. Computing the Types of the Relationships between Autonomous Systems. *IEEE/ACM Trans. on Networking*, 15(2):267–280, April 2007.

[25] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. *ACM/SIGCOMM*, Cambridge MA, USA, August 1999.

[26] V. Krishnamurthy, M. Faloutsos, M. Chrobak, J. H. Cui, L. Lao, and A. G. Percus. Sampling large internet topologies for simulation purposes. *Computer Networks, Elsevier*, 51(15):4284–4302, October 2007.

[27] D. Papadimitriou. Compact routing: Challenges, perspectives, and beyond. TRILOGY Future Internet Summer School 2009," Louvain-la-Neuve, Belgium, August 2009.

[28] A. Sprintson, M. Yannuzzi, A. Orda, and X. Masip-Bruin. Reliable Routing with QoS Guarantees for Multi-Domain IP/MPLS Networks. In *Proc. of IEEE INFOCOM 2007*, Anchorage, USA, May 2007.