λBGP: Rethinking BGP programmability

Nicholas Hart, Charalampos Rotsos, Vasileios Giotsas, Nicholas Race, David Hutchison

School of Computing and Communications, Lancaster University

***Abstract*—BGP has long been the de-facto control plane protocol for inter-network connectivity. Although initially designed to provide best-effort routing between ASes, the evolution of Internet services has created a demand for more complex control functionalities using the protocol. At the heart of this challenge lies the static nature of current BGP policy specification and enforcement, and the limited programmability of production BGP policy frameworks. Meanwhile, in other contexts, the SDN model has demonstrated that open and generic network control APIs can greatly improve network functionality and seamlessly enable greater flexibility in network management. In this paper, we argue that BGP speaking systems can and should provide an open control API and a richer policy language, in order to address modern era network control requirements. Towards this goal, we present λBGP, a modular and extensible BGP stack written in Haskell. The framework offers an extensible integration model for reactive BGP control that remain backwards compatible with existing BGP standards, and allows network managers to use a high-level language to define policy and inject dynamic information sources into path selection logic. Using kakapo, a high performance BGP traffic generator, we demonstrate that λBGP offers performance comparable to production BGP speakers while enabling complex AS route processing mechanisms in just a few lines of code.**

I. Introduction

The Border Gateway Protocol (BGP) is a routing protocol used by Autonomous Systems (ASes) to exchange routing and reachability information. The first version of the protocol was designed in the early days of the Internet and was intended to enable connectivity across the Internet. Although the protocol has undergone several revisions, version 4 [1] is the latest protocol version released in 1994, and the core functionality of the protocol has remained intact ever since.

Meanwhile, the requirements for the control plane of the Internet have been radically revised in the intervening years. The rate of growth in size and traffic in the Internet continues with no sign of reduction, and novel requirements for better than best-effort service delivery have emerged, while new control functions, like anycast, stress the policy mechanisms of the protocol. BGP currently faces significant challenges to maintain performance and reliability in the face of demands for highly adaptable and programmable inter-network connectivity.

At the heart of these challenges, we identify two major limitations. First, BGP configuration mechanisms have not evolved sufficiently to support complex policies. BGP configuration in current BGP routers consists of match-action rules with regular expression matching on path attributes. A user can use these rules to either filter paths from the RIB, bias the path selection algorithm of the BGP protocol, or supplement the information transmitted to adjacent ASes. Configuration is stateless and programmability is typically limited to simple arithmetic and logical expressions. As a result, network managers must maintain large and complex configuration files in order to achieve the required policy effects, requiring regular updates in order to support changing network conditions. Misconfiguration errors are a major cause of BGP problems [2], and connectivity outages and route leaks at a global or national level are not uncommon [3], [4].

Second, the BGP path selection process in current BGP routers is proactive, and its scope is limited to protocol information and static configuration data. Many measurement studies have highlighted the fact that BGP could improve the optimality and security of forwarding decisions by analyzing the information in the routing protocol or fusing routing information with external monitoring information [5], [6].

BGP policy mechanisms remain closed and lack support for run-time adaptation; as a result, human intervention is frequently required to adapt network policies to dynamic network conditions. Configuration automation can be achieved using scripts or distributed configuration platforms, such as Ansible [7], but unfortunately such make-or-break approaches are not sufficiently integrated to ensure correctness, and can lead to connectivity outages [8]. More recently, large content service providers have developed specialized BGP architectures that redesign the path selection mechanism to overcome protocol limitations, including edge peering scalability [9] and capacity-aware routing [6]. However, such solutions are very specific to the Internet edge context, and require extensive investment in hardware to support the required programmability in the network fabric.

In recent years, the rise of the Software Defined Networking (SDN) paradigm has highlighted the benefit of open control APIs in networking. SDN solutions have enabled a series of innovative and highly adaptive network functionalities in production networks in other application domains than IDR. BGP was an early target of SDN style programmability, with the development of platforms like RCP [10] and Morpheus [11]. Nonetheless, the direction of much of this work has been to deliver control centralization of legacy BGP routers, which reduces BGP resilience. More recently, research efforts have explored BGP as an east-west interface to their network control frameworks. Their design philosophy views BGP routing information as an additional policy input stream to optimize forwarding. As a result, implementations include minimal and monolithic BGP speaker implementations which lack support for several protocol features [12].

Enabling BGP programmability is critical in order to meet the growing control requirements of modern networks. BGP speakers require open and unified control APIs to enable suitable programmability, automation and security in inter-domain routing. We believe that the programming API should capitalize on the design philosophy of the protocol, respecting the underlying key design principles features which protect against routing instabilities at run time. Towards this goal, we present λBGP [13], a BGP stack written in Haskell, offering a flexible API for runtime BGP policy adaptation.

Specifically, we identify the following contributions:

- We present λBGP, an open-source BGP stack written in a functional language. λBGP offers a secure, performant and extensible BGP framework, that takes advantage of the static type system, high expressivity, and built-in concurrency and parallelization support in GHC/Haskell.
- We elaborate on the design of the BGP protocol, and define an open and flexible control API for BGP routers, which is compatible with existing BGP standards.
- We compare λBGP with existing production BGP implementations and identify that our system can deliver comparable processing performance for Internet-scale RIB tables. Our evaluation uses kakapo, an open-source high-performance BGP traffic generator and monitor written in C.

In the rest of this paper we first describe the functionality of the BGP protocol (§ II) and identify a set of functional requirements for a reactive BGP control API (§ III). Then we present the architecture of λBGP and discuss the design of its control API (§ IV). We next evaluate the performance of λBGP against popular production BGP speakers and demonstrate its flexibility through a reactive control application (§ V). Finally, we discuss related efforts in BGP programmability (§ VI) and conclude by discussing possible future directions (§ VIII).

II. BGP CONFIGURATION AND POLICY

All standards compliant BGP speakers operate within strictly defined constraints, designed to ensure routing stability and consistency at both inter-AS and intra-AS levels. The BGP protocol defines strict rules regarding the dissemination and integrity of protected route attributes. As long as it adheres to these protocol rules a BGP speaker has complete freedom to define its forwarding policy and the routes advertised to internal peers and other ASes. This flexibility is required for transit networks to operate securely and profitably. This routing policy license, and the mechanisms which embed it in a BGP router, find expression in the configuration languages implemented by all BGP speakers, in which the policy rules which can be expressed correspond closely to common business objectives.

BGP configuration primitives can be organized in four broad types.

Import filters, are applied before evaluating relative preferences between routes, and enable network managers to reject routes, from customers or peers, which are not known to be directly associated with those neighbours.

Export filters restrict routes advertisements to specific peers. Export filters are typically used towards provider networks to ensure that the traffic they send is destined to local AS customers.

Local route preference rules are used to rank preference of alternate viable routes to a specific prefix. Preference supplements the profit protection effects of outbound filters, typically to ensure that traffic is routed to the most profitable (or least costly) destination where multiple route options exist. Finer grained preference rules are also used to perform *traffic engineering*, e.g. to offload traffic at the nearest viable exit, or via the fattest or least congested pipes.

Finally, *policy tagging* rules, using the path attribute BGP community, allow network managers to mark routes to downstream BGP speakers in order to apply further filters or preference assignments. Generally, the net effect of BGP community tagging is an extension of the three aforementioned configuration primitives. For AS internal cases this may just simplify configuration, but in inter-AS cases it enables co-operative effects which would not otherwise be possible.

The realisation of the aforementioned configuration directives is realized through a three phase route processing and selection algorithm [1], depicted in Figure 1. *Phase 1* processes and filters and rank routes. Stage 1 is most relevant to eBGP routes, since iBGP routes already contain a Local Preference value from Stage 1 processing at the ingress border router, which would also be expected to filter unwanted routes. Phase 1 has a mandatory filtering rule which rejects any routes which contain the local AS number in the AS path

In *Phase 2*, route selection compares all available routes for each prefix, using Phase 1 ranking outcomes (*local preference*) selects, individually for each destination prefix, the highest ranking route. If more than one route ties for top rank then a fixed tie break algorithm is applied. Table I presents the route metrics used during the tie break process, in priority order. The operationally most significant metrics are the path length, the source of the route (eBGP over iBGP) and IGP metrics. Another path attribute, MED, allows a directly connected external AS to signal preference when multiple alternate links are available.

Phase 3 implements peer specific filters over routes which won out in phase 2. A mandatory Stage 3 rule excludes routes *received* from internal peers from onward forwarding to further internal peers. Although not directly relevant to route selection, Stage 1 and Stage 3 also allow path attributes to be modified, subject to certain rules. There are some mandatory path attribute modification rules: when sending to an *external* peer the received AS path list attribute is extended with the local AS number, and the local preference attribute removed. BGP speakers widely apply tagging of routes, using a flexible path attribute named BGP Community. Stage 1 and 3 processes can arbitrarily read and modify community tags and use them in their filtering and ranking functions.

The BGP protocol offers two main points for policy enforcement. Firstly, during phase 1, the network manager can assign preference to paths and affect route filtering and selection,

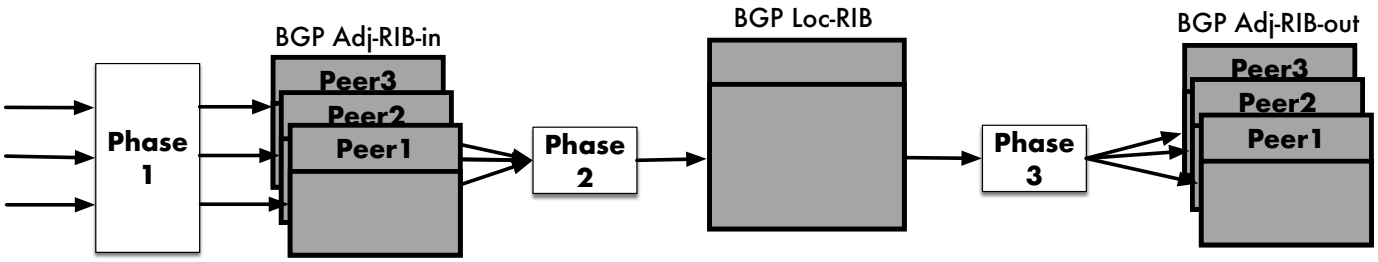


Figure 1: Path processing in BGP. Forwarding state in split into per-session Adj RIB-in and Adj RIB-out structures and a global LOC-RIB. Each route update is process by a three phase route validation and selection process.

Path metric	Description
Local Preference	Network-wide path preference
AS_PATH length	Route AS hop count
Origin	Binary valued attribute set at route origin
MED	Neighbour AS signalled link preference
EBGP	Prefer routes received from external peers
IGP metric	IGP metric to next hop
Peer Router ID	Arbitrary criterion
Peer address	Arbitrary criterion

Table I: Phase 2 route selection criteria, ordered on precedence. Selection criteria in bold are most significant.

before their insertion in the RIB. Secondly, the network manager can manipulate route communities or filter router in phase 3 and effectively control route advertisements to peering ASes.

Compliance with the protocol-specified configuration points is essential if effective BGP programmability is to maintain routing stability and consistency.

Safeguarding against AS internal forwarding loops was an important design objective in the original BGP architecture, and the specified sequencing of filters and preference calculation *before* insertion into a common RIB implicitly enforces the important rule that the same local preference should be advertised to all internal peers and the maintenance of consistency in local preference values across the AS.

Equally important in this regard, but only a convention, is the principle that route filters are only applied on BGP sessions with external peers. However, there is no restriction on changing (and re announcing) calculated preference or other path attributes for an existing route: in conventional routers this rarely happens, but it can. soft reconfiguration [14] allows route policy to be dynamically reconfigured, after which existing received routes are reevaluated and the resulting routes re-advertised, which may be entirely different to those previously announced. Thus we can see that dynamic policy is not a new, or questionable concept, and also that RFC based constraints are insufficient to protect against fundamental risks in applying policy in BGP networks.

III. BGP PROGRAMMABILITY CHALLENGES

In the recent years, the functional requirements for the BGP protocol have changed drastically. The number of Internet peers has increased exponentially, introducing scalability challenges for FIB sizes and the protocol convergence times [15]. In parallel, BGP policy mechanisms are required to enable a whole array of new control plane functionalities, like anycast forwarding [16], security and capacity-aware load-balancing [6].

Existing configuration paradigms are quite effective for expressing many network policy requirements, especially those which optimise long terms commercial objectives. But routing policies in support of security and service protection are difficult or impossible to encode in static configuration languages. Examples are responses to inadvertent or malicious routing failures, or responding to dynamic network conditions, e.g. congestion, route instability, or sub-optimal routing from a preferred peer. For these cases we require programmability, and in many case access to persistent and/or external state. These requirements motivate the development of open, programmable, BGP speakers.

In order to initiate a discussion on the programmability requirements of modern networks with respect to BGP, we identify two specific use-cases and discuss the challenges they raise.

Security:: A common security challenge for BGP networks is Path hijacks, the announcement from a malicious AS of paths that place the AS as the best route in an effort to eavesdrop or damage service delivery. BGPSec is a protocol extension enabling ASes to cryptographically sign and validate route updates using a Resource Public Key Infrastructure (RPKI). Unfortunately, BGPSec support remains limited across the Internet ($\sim$16% of total routes have valid certificates) [17], exposing ASes to path hijacking. The research community has proposed numerous mechanisms to include additional information in the path selection process [18], [19]. Nonetheless, existing BGP policy mechanisms remain proactive and lack automation.

Performance-aware routing:: Internet traffic is increasingly shifting toward delay-sensitive applications such as video streaming, VoIP, gaming and SaaS [20]. Fundamental changes in the Internet interconnection strategies led to the flattening of the topology to bring content providers closer to "eyeball" ASes [21], but the inability of BGP to incorporate performance-aware metrics creates significant challenges in meeting performance demands [22], [23], [24]. BGP traffic engineering policies lack predictability and responsiveness, and

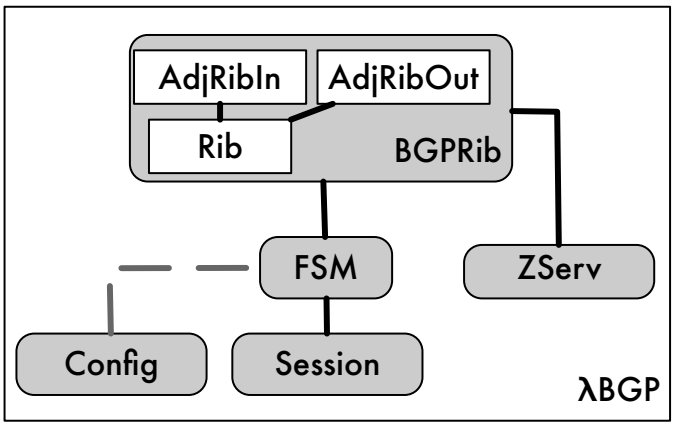


Figure 2: λBGP architecture.

often do not bring the desired results [25], [26], while existing proposals either require protocol changes that are unlikely to be adopted [27], or they rely on data-plane "optimizers" that are prone to instabilities and misconfigurations [28], [29].

IV. λBGP ARCHITECTURE

A. A Route Policy API

Section 2 motivates the design approach for a dynamic, programmable BGP speaker, which is nonetheless RFC compliant. At its lowest level, the programmable BGP speaker provides a simple callback API, which is invoked whenever route Updates are received, and again whenever a change to the main RIB occurs as a result of a new route or route withdrawal. These call-outs correspond exactly to the Stage 1 and Stage 3 processing phases for the BGP route selection process as described in RFC4271. This low-level API also has the capability to re-evaluate routes received in the past and trigger routing changes asynchronously to routing update message reception. An important aspect of this API is that it effectively guarantees RFC compliance, as long as it adheres to the basic principles that it does not reference other route state whilst processing, and that the path attribute changes it makes are permitted by the applicable RFCs. The earlier discussion on soft-reconfiguration points to two variants on this API: a simpler version which is only capable of reacting to routes on arrival, and a second, more powerful version which mimics soft-reconfiguration.

B. λBGP Design

```
evalLocalPref :: PeerData -> [PathAttribute]
-> [Prefix] -> IO Word32
updateCommunities :: PeerData -> [PathAttribute]
-> [Prefix] -> Maybe [PathAttribute]
filterExport :: PeerData -> [PathAttribute]
-> [Prefix] -> Bool
```

Listing 1: Signature of the API callback functions which allow users to compute run-time local preference values, manipulate communities and filter route advertisements.

Our BGP implementation aims to fulfill three key design goals. First, we developed a BGP speaker in a functional programming language, to improve the security and flexibility of the code. The abstraction and static type checking of functional languages can eliminate a large portion of coding errors , while the brevity of the language reduces code size and potentially the attack surface. Second, we define a control API for BGP speakers supporting evolutionary programmability, backwards compatible with existing standards. The proposed API allow a network manager to interject in the route processing algorithm defined in the BGP standards and at run-time adapt the forwarding policy. Third, our API enables a reactive control API, capable of incorporating multiple information sources during route selection.

λBGP [13] is a complete implementation of a BGPv4 router in Haskell. The source code of λBGP is open source, under the Apache 2.0 license. The router offers a complete implementations of all the components of a BGP router, including the finite state machine and the RIB. In addition, our implementation supports BGP, MRT, and BMP parsing and Zserv integration, to deploy FIB updates on the data plane through the zebra service.

λBGP is a fully-fledged BGP router which enforces all of the explicit and implied rules in RFC4271, whilst enabling conforming programmable behavior. These rules are essential to prevent a misbehaving policy 'program' from causing unpredictable or catastrophic disruption to connected networks.

The λBGP architecture is depicted in Figure 2 and consists of 4 modules. The *session* module manages network-level connectivity with adjacent BGP peers and serializes/deserializes BGP messages. The *FSM* module implements a BGP finite state machine and manages the BGP session lifecycle. Once a BGP session is established, BGP messages are passed to the *BGPRib* module, which processes BGP update messages into BGP RIB manipulations. The BGPRib is responsible for deploying FIB updates, using the *Zserv* module, and generates route updates to adjacent peers.

It is worth highlighting that the modular design of λBGP enables easy modification of different functionalities of a BGP speaker. For example, the source code λBGP offers two RIB module implementations. The one used in the evaluation section, employs the IntMap data structure of the Haskell runtime to store route data. Additionally, the source code contains an alternative RIB implementation that uses a compressed data structure to explores performance gains achieved due to route groupings in the Internet [30]. Similarly, the code offers alternative Session module implementations, that allow the use of different parsers (e.g. MRT, BMP) to feed route updates to λBGP.

λBGP offers a programmable and reactive control API. A network manager can implement route processing logic in Haskell and inject them in the main processing logic of the BGP algorithm. Nonetheless, to ensure conformance with BGP standards, our framework enables programmability in two primary phases: during the computation of the local preference and the communities of a route, and the filtering of a

route advertisement. Listing 1 provides the function signatures of the respective callbacks. The `evalLocalPref` function implements a custom local preference calculation algorithm. The `updateCommunities` function allows manipulation of the communities attribute of a route. The `filterExport` function allows router filter on the Adj-RIB out, effectively controlling which routes are exported to each peer. It is important to point out that the evalLocalPref is an IO action function, which allows non-pure function implementations, performing IO operations and accessing persistent state.

If a local preference calculation performs external IO, its processing and subsequent advertisement can be delayed. This can impact negatively the convergence and responsiveness of the protocol. To address this challenge, the λBGP API offers the optional runtime capability of an asynchronous processing mechanism, on a per-route basis. The *asynchronous route processing* mechanism allows the BGP speaker to further process a route update after the initial calculation of the local preference value is completed. The *asynchronous route processing* mechanism processes a route update using initially a simple default algorithm. In parallel, the BGPRib module delegates the computation of the local preference, using the enhanced function, to an asynchronous thread. When this thread completes, and if the re-calculated local preference has changed, then the BGPrib module will re-process the route and disseminated any required updated route announcements. In order to avoid processing of stale updates, the router assigns a monotonically increasing id for each route update for a given prefix. If a route that is currently expecting the computation of its local preference is overwritten by a new update, then the thread is canceled and the asynchronous route selection terminated. Whilst this strategy does have the potential to generate increased routing traffic if applied too freely, the asynchronous route evaluation algorithm can be designed to mitigate this problem by ensuring that only highly ranked threats are actioned immediately, and if required, rate limiting its own actions. Otherwise the asynchronous processing function can simply alert network managers.

V. Evaluation

In this section we evaluate the performance of our system and demonstrate the flexibility using an advanced BGP policy configuration. In summary, λBGP achieves comparable processing overheads against production software BGP speakers even when processing large route tables. In parallel, the design of λBGP allows the realization of highly adaptive policy mechanisms, which cannot be implement using existing BGP policy and configuration tools.

A. λBGP performance

To evaluate the performance of λBGP, we developed *kakapo* [31], a BGP traffic generator and logging framework written in C. Kakapo implements a minimal BGP speaker, which allows BGP session establishment, BGP update message generation and logging. The tool is designed to achieve high traffic throughput and logging precision. It uses BGP packet pre-crafting and socket `IOVECS`, to minimize the per-packet processing latency, while the user can control the size, content and frequency of BGP update bursts in addition to the overall size of route table.

To ensure consistently unique routes in repeated updates, routes are varied by changing intermediate AS numbers in the AS path. For each BGP speaker, we configure two peering sessions to the kakapo tester, on different network addresses. One session is used to inject a stream of BGP updates to

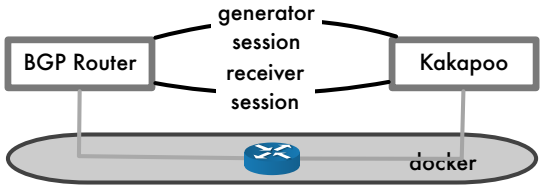
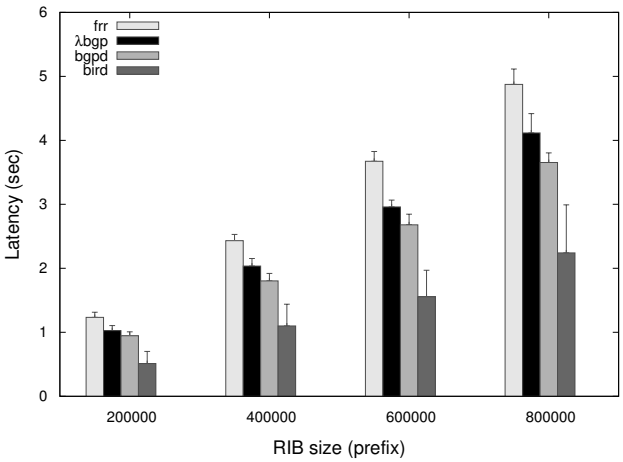
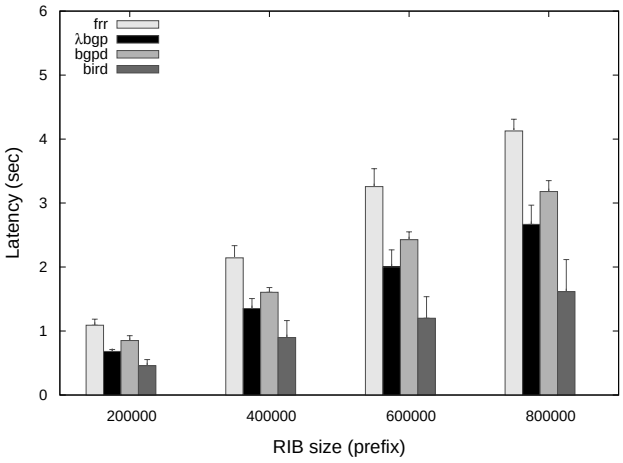


Figure 3: Experimental topology



(a) 5 routes per update



(b) 10 routes per update

Figure 4: Performance comparison of λBGP against production BGP speakers, which significantly improves as more prefixes are included in an update.

the speaker, while the second session logs the time it takes for the BGP speaker to consume, process and re-advertise each update. We define *processing latency* as the time period between the transmission of the first update, in a burst of updates on the source BGP session, to the receipt in the logging peer of the last update in the burst.

For this measurement we employ the topology depicted in Figure 3. In the experiments we use a Dell R630 server, hosting KVM virtualised containers and using the Docker daemon for orchestration. OS is Centos 7; Docker version 18.09; Haskell is GHC version 8.6.5. Both BGP speakers and the kakapo tester run as docker instances in separate VMs; virtualised point-to-point links are used to connect VM hosts.

Our BGP performance testbed measures many aspects of routing performance, including continuous throughput and single update processing latency, generated either by a single or multiple concurrent source BGP speakers. However, some metrics are more sensitive to specific experimental conditions than others, or exhibit distracting anomalies. Therefore in this paper we present experimental results for one readily understandable performance metric, which is broadly consistent with other metrics we have measured. This metric is the time taken to fully process and re-announce single large route tables. This processing function correlates with the real-world scenario following the establishment of a new BGP peer session.

We define two control parameters for the experiments in this report. The first control parameter is the overall size of the routing table (RIB), the second is the number of Update messages used to convey a complete route table. The route table is synthetic, and contains up to 800,000 distinct prefixes, uniformly announced as 5 prefixes per Update, for a total of 160,000 distinct origin ASes and AS-paths. This size of route table and ratio of paths to prefixes is representative of the observed data in contemporary RIPE RIS and RouteView datasets, and thus corresponds closely with operational reality for a core transit border router.

In our experiments we compare λBGP against three production BGP speakers: BIRD[32], FRR [33] and openBGP [34]. Each experiment is repeated 10 times and we present in figure 4 the mean for each update size.

The first graph show that as the size of the route table increases the relative performance of the various BGP speakers remains constant, and also that there is a roughly linear increase in elapsed time as the route table size increases (it is not the purpose of the present paper to analyze in detail the scaling behavior of BGP speakers). The graph shows that there is a range in performance between the BGP speakers of around 2:1, and that bird2 is the best performer. FRR is the slowest: openBGP and hBGP are 2nd and 3rd respectively. The significance of the second graph is that by concentrating identical size route tables over smaller numbers of paths and thus also a smaller number of Update messages, there is an improvement in performance of all BGP speakers. This is to be expected, since some part of the total processing effort is devoted to the mechanics of processing protocol messages, for example in the parsing process: also, there may be a smaller number of user/operating system interactions, because of the smaller number of messages and reduced size of the TCP data stream. The noteworthy observation is that in this second scenario the observed relative performance order changes, and the Haskell implementation rises to second place. From this we conclude that the current Haskell implementation is a little less optimized for the tasks of parsing messages, but performs better when it comes to accessing and updating the large data structures which compose BGP RIBs. The overall conclusion however is that a relatively naive implementation in Haskell is capable of delivering comparable or even better control performance than well written C language equivalents.

There are of course caveats: in these examples both the native C BGP speakers and the Haskell speaker are applying minimal policy logic to route selection, although they must all run a full LocRIB / Phase 2 best route selection process. The question of how the performance ordering would change in the presence of complex policy is one for further study, however the fact that Haskell implementation enables the relevant processing to be taken in parallel, and the possible inefficiency of the C language speakers implementation of configuration language statements, compared with the possibility of a compiled, optimized, equivalent in Haskell, will make for interesting further work.

B. λBGP programmability

```
type AdjacencyMatrix =
    Data.IntMap.IntMap ( Data.IntMap.IntMap ())

adjCheckPair :: AdjacencyMatrix -> Int -> Int -> Bool
adjCheckPair adjMap a b =
    maybe False ( Data.IntMap.member a)
     ( Data.IntMap.lookup b adjMap )

adjcheckPath :: AdjacencyMatrix -> [Int] -> Bool
adjCheckPath m (a:b:zx) = adjCheckPair m a b
                          && adjCheckPath m ( b:zx )
adjcheckPath _ _ = True

evalLocalPref :: PeerData -> [PathAttribute]
                 -> [Prefix] -> IO Word32
evalLocalPref peerData pathAttributes _ = do
    adjMatrix <- adjMatrixFromPeer peerData
    let asPath = getASPath pathAttributes
        validPath = adjCheckPath adjMatrix asPath
    return $ if validPath then peerLocalPref peerData
                          else 0
```

Listing 2: A sample local preference calculation function which reduces the local preference value if the path contains unknown AS adjacencies.

To evaluate the effectiveness of the λBGP API, we present in this section the implementation of a policy that deprecates paths containing previously unseen AS connections. The underlying idea is that BGP misconfigurations or prefix hijacks manifest as advertisements of bogus AS paths, which may be detected by constructing an AS adjacency matrix from globally

collected route advertisements. The policy could be applied at the level of adjacent AS pairs, or longer sequences; however, in this example we address just the first case. Detection of such "adjacency anomalies" has been shown to be effective in preventing traffic misdirection attacks [35], but have not been implemented in practice due to the limited programmability of the existing BGP implementations.

Listing 2 presents a sample function that consumes a pre-calculated AS adjacency matrix, and monitors whether connectivity of any AS pair has been observed previously. Also, our implementation uses an asynchronous local preference calculation method to monitor the path over time and update the local preference accordingly. Specifically, if adjacency between a pair of ASes is seen for the first time, then the function reports the new path to an external server, delays a short time, and refreshes the adjacency check database. Thus the initial route update is processed immediately, but if the external server determines that the AS path is invalid, the path may be demoted, and most likely removed from the route table. It is important to mention that no comparable functionality is currently supported by any production BGP router. It is notable that the LOCAL_PREF evaluation call-out is able to return multiple "live" results. Specifically, a continuation function is passed into the user-provided evaluation function, which may then return an initial result in a timely fashion, but then later generate additional differing results, potentially triggering further route updates and re-advertisement, if appropriate.

VI. Related Work

The research community has explored a series of approaches to enable BGP programmability, which predominantly rely on separating the routing logic from the routers. Of great relevance to our work is XORP [36], an open-source extensible multi-protocol routing platform enabling integration of IGP and EGP protocols. The platform offered a policy language for the configuration of individual routing protocols and the management of information exposure between protocols. Although the policy language offered a number of novel features, such as the ability to influence EGP route advertisement based on IGP state, it was not Turing-complete. ExaBGP [12] is a popular BGP speaker equipped with a REST configuration API for route injection. ExaBGP is used extensively for rapid prototyping of BGP experiments, as well as to expose BGP control to network applications. In contrast to λBGP however, ExaBGP is a single-threaded Python system, lacking a complete BGP speaker/RIB/route selection engine and is optimized as a responsive offline tool rather than a full fledged active element in an iBGP mesh.

Research on BGP has often proposed logically centralized control architectures of BGP networks. Routing Control Platform (RCP) [10] proposed the introduction of BGP programmability across a network by decoupling the routing state from the BGP routers and introducing a centralized route control server. RCP reduces the complexity of fully-distributed path computation by introducing a centralized Route Reflector, which performs route selection based on IGP routing information and the network topology. Unfortunately, the EGP visibility of the route control server is hindered by the adoption of iBGP, which dictates that edge routers will expose only the best route for a each prefix. To address these challenges Verkaik *et al*. [37] developed IRSCP, an RPC extension with support for high-availability and fine-grain BGP policy. Control applications can define an explicit egress ranking per prefix, enforced by the control infrastructure at run-time. Similar to IRSCP, Morpheus [11] proposes outsourcing the BGP routing selection process to a small selection of servers which offer programmable multi-criteria route selection. Morpheus supports multiple route classification mechanisms, combined using a weight-based approach, and per-flow state persistence. These authors all resolve the issue of gaining visibility to external routing state by either intercepting or duplicating all external peer BGP sessions, and do not provide a programmable online routing solution, relying instead on pre-computed configuration with a non-realtime refresh capability.

We believe that the design of λBGP offers an extensible platform which overcomes these limitations to implement and extend existing centralized BGP control architectures and expand BGP programmability across the network.

More recent proposals advocate SDN-based centralized control planes to improve network management, the expressiveness and flexibility of traffic engineering policies, and convergence time [38], [39], [40]. The SDN-IP [41] application introduces support for iBGP in the ONOS platform and explore the applicability of BGP as an East-West Interface. Software Defined Internet eXchanges (SDXs) [42], [43] aim to address some inherent limitations of BGP in the context of IXP route servers, which enable very dense peering connectivity that can therefore provide the significant benefit of finer-grained routing policies. Espresso [9] and Edge Fabric [6] introduce novel BGP control architectures that centralized route control and enable novel criteria in route selection, including path performance and capacity. All of these approaches are orthogonal to λBGP, since they bypass the conventional BGP internal mesh entirely to implement programmable routing logic, whilst λBGP is designed to integrate synergistically within the existing iBGP routing mesh.

VII. Discussion and Future Work

The proposed control API is only a first step towards enabling appropriate BGP programmability. Nonetheless, even the simple illustrative code fragment in Listing 2 would actually have correctly detected and deprecated the route leaks in the case of the Verizon outage of 20019[8].

It is important to note that this API is the lowest level API possible: higher-level, application specific, APIs, such as an implementation of a policy DSL, may be built using this API, and in fact that is our main purpose in providing this low-level API. However, even this API is sufficient to implement useful policies which are not otherwise possible in conventional BGP speakers. A more advanced API implementation might

also choose to provide composable policy, or dynamically reloadable policy modules - alternatively, an implementation might outsource the task of route evaluation to an external system after making an initial fast response based on conventional static configuration, with the option of overriding an initial evaluation response. These options are orthogonal to the topic of developing high-level policy APIs, but the concept of composing security, performance and business policy as distinct implementations is an obvious direction for future investigation, and the mechanisms described would allow different approaches for different classes of policy. A concrete example would be a dynamic security module which would allow mitigation for specific newly emerging threats to be deployed, composed with a static business policy, *and* an externally moderated traffic engineering system which can update granular route selection in response to changing network loads.

We believe that new BGP architectures can be developed around such λBGP instances to provide AS-wide programmability and enable support in legacy hardware routers. Connecting λBGP through iBGP to AS local routers can enable the manager to bias the route selection of already deployed legacy border routers in the network. Nonetheless, designing such architectures requires extensive experimentation to understand the trade-offs of different configuration strategies. For example, if standard iBGP is used to connect legacy BGP routers to λBGP, then programmability may be limited due to restricted network visibility, since legacy routers typically only advertise their selected routes rather than the complete BGP message feed. However, increasing deployment off BGP_ADDPATH and BMP capability offers the prospect of improved visibility over diverse routes at AS borders.

Finally, we believe that λBGP can provide an ideal environment to explore reactive BGP policy verification. The network community has developed multiple BGP policy verification frameworks [44], [45] offering high-level intent languages and static analysis frameworks to validate BGP policies. Such mechanisms operate in an offline fashion and allow network managers to detect misconfigurations prior to deployment. The development of λBGP using a functional language and its open control API allow integration of similar mechanisms in the heart of the path selection algorithm. In parallel, the support of λBGP for asynchronous path computation can amortize the computational costs of static analysis, while the verification can include live network state to improve results.

VIII. Conclusion

In recent years, Internet evolution has introduced a requirement for greater reactivity and flexibility in the control plane of the network, in order to fulfill the performance and availability demands of many modern network services. Unfortunately, BGP policy frameworks remain static, closed-loop and reliant on low-level policy languages that lack support for appropriate programmability. In this paper, we revisit the implementation of BGP systems and present an open control API for BGP routers and route controllers that ensures conformance with protocol standards. More significantly, we present λBGP, an implementation of a BGP router written in Haskell that implements the proposed control API. An essential attribute of the specific API described is the enforcement of BGP protocol rules over the range of actions available to API clients, which contrasts with the unlimited and potentially dangerous impact of more generic programmable BGP frameworks. Furthermore, the described implementation combines programmability with performance at Internet scale, and the experimental results show that there need be no trade-off between these contrasting goals. Using a novel BGP testing framework, we demonstrate that our implementation can achieve performance comparable with production BGP implementations. Additionally, we demonstrate the expressiveness of the proposed solution using an example local preference calculation method which validates new routes against a dynamically maintained AS adjacency matrix.

Acknowledgments

The authors are grateful to the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the TOUCAN (EP/L020009/1) and INITIATE (EP/P003974/1) projects, and EPSRC and British Telecoms (BT) for funding the NG-CDI (EP/R004935/1) project, which supported much of the work presented in this paper.

References

[1] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," Internet Requests for Comments, RFC Editor, RFC 4271, January 2006, http://www.rfc-editor.org/rfc/rfc4271.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4271.txt

[2] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: ACM, 2002, pp. 3–16. [Online]. Available: http://doi.acm.org/10.1145/633025.633027

[3] A. Robachevsky, "Google leaked prefixes and knocked Japan off the Internet," https://www.internetsociety.org/blog/2017/08/google-leaked-prefixes-knocked-japan-off-internet/, Aug. 2017.

[4] RIPE, "YouTube Hijacking: A RIPE NCC RIS case study," https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study, 2008.

[5] V. Giotsas, G. Smaragdakis, C. Dietzel, P. Richter, A. Feldmann, and A. Berger, "Inferring bgp blackholing activity in the internet," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131379

[6] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 418–431. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098853

[7] "Ansible," https://www.ansible.com/, 2019.

[8] T. Strickx, "How verizon and a bgp optimizer knocked large parts of the internet offline today," http://tiny.cc/v2o29y/, 2019.

[9] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*,

ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 432–445. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098854

[10] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 15–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251205

[11] Y. Wang, I. Avramopoulos, and J. Rexford, "Morpheus: Making routing programmable," in *Proceedings of the 2007 SIGCOMM workshop on Internet network management*. ACM, 2007, pp. 285–286.

[12] Exa Networks, "exaBGP," https://github.com/Exa-Networks/exabgp, 2019.

[13] "hBGP - A complete functional language implementation of a BGP stack (BGP speaker, dataplane, MRT and BMP)," https://github.com/hdb3/hbgp/, 2019.

[14] E. Chen, "Route refresh capability for BGP-4," Internet Requests for Comments, RFC Editor, Tech. Rep. 2918, September 2000. [Online]. Available: rfc2918

[15] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever, "Swift: Predictive fast reroute," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 460–473. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098856

[16] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, "Internet anycast: Performance, problems, & potential," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 59–73. [Online]. Available: http://doi.acm.org/10.1145/3230543.3230547

[17] N. I. of Standards and Technology, "Global prefix/origin validation using rpki," https://rpki-monitor.antd.nist.gov/, 2019.

[18] M. Konte, R. Perdisci, and N. Feamster, "Aswatch: An as reputation system to expose bulletproof hosting ases," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 625–638, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787494

[19] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "Artemis: Neutralizing bgp hijacking within a minute," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2471–2486, Dec. 2018. [Online]. Available: https://doi.org/10.1109/TNET.2018.2869798

[20] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 75–86, 2011.

[21] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "The flattening internet topology: Natural evolution, unsightly barnacles or contrived collapse?" in *International Conference on Passive and Active Network Measurement*. Springer, 2008, pp. 1–10.

[22] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?!: it must be bgp," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 75–84, 2007.

[23] R. Bian, S. Hao, H. Wang, A. Dhamdere, A. Dainotti, and C. Cotton, "Towards passive analysis of anycast in global routing: Unintended impact of remote peering," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 3, 2019.

[24] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, "Internet anycast: performance, problems, & potential." in *SIGCOMM*, 2018, pp. 59–73.

[25] B. Quoitin, C. Pelsser, O. Bonaventure, and S. Uhlig, "A performance evaluation of bgp-based traffic engineering," *International journal of network management*, vol. 15, no. 3, pp. 177–191, 2005.

[26] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[27] R. Oliveira, M. Lad, B. Zhang, and L. Zhang, "Geographically informed inter-domain routing," in *2007 IEEE International Conference on Network Protocols*. IEEE, 2007, pp. 103–112.

[28] M. Luckie, "Spurious routes in public bgp data," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 14–21, 2014.

[29] M. Levy, "The deep-dive into how verizon and a bgp optimizer knocked large parts of the internet offline monday," Cloudflare Blog. https://tiny.cc/hjm29y, July 2019.

[30] A. Broido and k. claffy, "Analysis of RouteViews BGP data: policy atoms," in *Network Resource Data Management Workshop*, Santa Barbara, CA, May 2001.

[31] "kakapo - A BGP traffic flood tool - source, sink and monitor, + analytics," https://github.com/hdb3/kagu, 2019.

[32] CZ.NIC Labs, "The BIRD Internet Routing Daemon," https://bird.network.cz/, 2019.

[33] Linux Foundation, "FRRouting," https://frrouting.org/, 2019.

[34] "OpenBGPD," http://www.openbgpd.org/, 2019.

[35] Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Argus: An accurate and agile system to detecting ip prefix hijacking," in *2011 19th IEEE International Conference on Network Protocols*. IEEE, 2011, pp. 43–48.

[36] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov, "Designing extensible ip router software," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 189–202.

[37] P. Verkaik, D. Pei, T. Scholl, A. Shaikh, A. C. Snoeren, and J. E. van der Merwe, "Wresting control from bgp: Scalable fine-grained route control," in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ser. ATC'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 23:1–23:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364385.1364408

[38] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: better internet routing based on sdn principles," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 55–60.

[39] P. W. Thai and J. C. De Oliveira, "Decoupling bgp policy from routing with programmable reactive policy control," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 47–48.

[40] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 13–18.

[41] SDN-IP Architecture, "ONOS," https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture, 2019.

[42] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined internet exchange," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 551–562.

[43] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 1–14.

[44] N. Feamster, "Practical verification techniques for wide-area routing," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 87–92, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/972374.972390

[45] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Scalable verification of border gateway protocol configurations with an smt solver," *SIGPLAN Not.*, vol. 51, no. 10, pp. 765–780, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/3022671.2984012