Towards Logically Centralized Interdomain Routing
Shahrooz Pouryousef, Lixin Gao, and Arun Venkataramani,
University of Massachusetts at Amherst
https://www.usenix.org/conference/nsdi20/presentation/pouryousef
Towards Logically Centralized Interdomain Routing

Shahrooz Pouryousef, Lixin Gao, and Arun Venkataramani

University of Massachusetts Amherst

Abstract

In this paper, we present the design and implementation of CIRCA, a logically centralized architecture and system for interdomain routing that enables operators to offload BGP-style route computation to the cloud while preserving the confidentiality of proprietary information. To this end, our work presents the first provably safe, live, and fully distributed convergence detection algorithm for decentralized policy routing and, somewhat surprisingly, shows that long MRAI timers can likely be completely eliminated while significantly improving convergence delays with logical centralization. Our experiments with a Quagga-based CIRCA prototype and the Internet’s AS topologies suggest that CIRCA can improve interdomain routing convergence delays and transient route inconsistencies by over an order of magnitude and offers non-trivial incremental deployability benefits with modest changes to widely deployed routing infrastructure.

1 Introduction

Logical centralization of control and management for enterprise networks has proven successful in recent years. However, this trend has minimally, if at all, affected interdomain routing in the Internet that has seen little fundamental change in over two decades of operation and continues to suffer from long convergence delays, lack of control-data separation, poor management knobs, lack of evolvability, etc.

Logical centralization or cloud-assisted interdomain route computation holds the promise of alleviating these longstanding problems but is not easy to accomplish. Unlike enterprise networks, a key stumbling block is the need to maintain the confidentiality of proprietary routing policies. Recent research [2, 17] has attempted to attack this problem by employing secure multiparty computation, but its inherently computationally-expensive nature poses a scaling challenge, so it has been demonstrated only at small scales with restrictive assumptions on the expressiveness of routing policies. Cloud-assisted interdomain route computation has therefore been limited to narrower contexts such as software-defined exchange points [15, 16], engineering traffic across multiple ASes owned by a single enterprise [18, 53], or compromising on either or both of scale and policy confidentiality [26].

In this paper, we present the design and implementation of CIRCA, the first logically centralized interdomain routing control architecture and system enabling operators to offload BGP-style route computation as-is to the cloud while preserving the confidentiality of proprietary policies. The CIRCA service makes the following assurances to network operators: (1) forwarding entries computed by CIRCA will be equivalent (as formalized in §3.4.1) to what their routers are computing today; (2) CIRCA will quickly return forwarding entries reflecting a converged state of the network circumventing BGP’s long tail of convergence delays (tens of seconds to minutes); and (3) CIRCA does not require an AS to disclose proprietary policy information to any third party except to the extent that it can already be inferred by its eBGP peers.

The high-level design of the baseline CIRCA system, as illustrated in Figure 1, is simple: each router speaking BGP (or ground router) has a virtual router incarnate (or avatar) in the CIRCA cloud. Upon detecting a root cause event, a ground router dispatches it cloudward to its avatar; the virtual routers in the cloud thereafter simply replay the same BGP protocol as ground routers but in a significantly “fast-forwarded” manner; and upon convergence, cloud routers dispatch modified forwarding entries to their ground incarnates.

A natural question is why this simple, perhaps even seem-
ingly naive, high-level design can enable the cloud to compute routes faster than the ground if it’s replaying the same protocol. The answer is threefold: (1) the cloud can easily afford orders of magnitude more control bandwidth, e.g., tens or even hundreds of Gbps of LAN bandwidth, compared to the ground; (2) propagation delays in the cloud (< 1 ms or even just a few microseconds [21, 33, 35, 55]) are several orders of magnitude lower than the delay diameter of the Internet (≈ hundreds of ms); (3) the (cloud) control plane is physically isolated from and cannot interfere with the (ground) data plane. Thus, the cloud has the luxury of doing away with long route advertisement timers that in ground-BGP today are believed necessary to mitigate the risk of super-exponential message complexity [9, 28, 29, 36, 37] and conservatively set to high values (e.g., 30s is a common vendor default).

The deceptively simple exposition above hides a number of research questions that must be answered to translate the high-level design to a deployable system in practice. Can the cloud really compute stable routing outcomes an order of magnitude faster than the ground? Can (and how?) cloud routers quickly detect that the distributed processing of a root event has converged? Can CIRCA guarantee consistency of computed routes and ensure that its computed routing outcomes will match those of BGP despite component or network failures? Can CIRCA coexist with BGP and is it incrementally deployable for incremental benefit?

Tackling the above questions and answering them in the affirmative aided by an implemented prototype of CIRCA is our primary contribution comprising the following parts:

1. Distributed convergence detection: Design and implementation of the first fully distributed BGP convergence detector that is provably safe, i.e., no false positives, and live, i.e., detection incurs a modest bounded delay (§3.3).

2. Quick end-to-end convergence: Large-scale prototype-driven experiments showing that CIRCA can ensure predictably quick convergence reducing BGP’s tail convergence delays by over an order of magnitude, in part by eliminating unnecessary long timers (§3.2, §4.1).

3. Route computation equivalence: A design that provably ensures that its computed routing outcomes are equivalent to those of any convergent (formalized in §3.4) distributed policy-based ground routing protocol.

4. Incremental deployability: Design and evaluation of mechanisms to deploy CIRCA co-existent (§3.5.1) with BGP as well as in an incremental manner (§3.5.2).

2 Background and lineage

Logical centralization of network control and management, including for interdomain routing, has a long scientific lineage. We overview what prior work has accomplished on that front to position how CIRCA builds upon that work.

In intradomain routing, a line of work seemingly starting with calls for "routing as a service" [30] or "separating routing from routers" followed by works such as RCP [4], 4D [52], Ethane [5] etc. spurred what is now known as software-defined networking (SDN), a term that commonly means logical centralization of control and management for enterprise networks. Much follow-on SDN research as well as widespread embrace by industry firmly attests to its benefits such as cleaner control and data separation, ease of management with a global view, hardware-software decoupling, etc.

Interdomain routing on the other hand, since BGP’s creation in the late 80s and progressive standardization through the 90s, has remained largely untouched by the logical centralization trend as BGP4 continues to be the de facto interdomain routing protocol. Recent research however has begun to explore extending SDN approaches to the interdomain context [1, 41, 44, 54]. For example, SDX by Gupta et al. [16] is a software-defined Internet exchange point (IXP) that enables more sophisticated packet processing rules to engineer interdomain traffic. Kotronis et al. [10, 24–27] also advocate a logically centralized “multi-domain SDN” approach as an alternative to BGP, introduce a publicly available (single-node) Mininet-based emulation platform for experimenting with hybrid BGP-SDN routing, and show modest improvements in convergence delays because of centralization. Gupta et al. [2, 17] advocate using secure multiparty computation (SMPC) so as to combine the benefits of logical centralization with confidentiality of proprietary routing policies, and argue that SMPC may be computationally feasible for BGP with some policy restrictions. Recent research has developed approaches for preserving the privacy of ISP policies at Internet exchange points [6, 7, 14].

Consensus routing [20] advocates a consistency-first (in contrast to the Internet’s longstanding allegiance to soft-state) approach to interdomain routing. It works by periodically capturing a consistent snapshot of global network state—e.g., by having a small number of say tier-1 ASes engage in a consensus protocol to agree upon globally in-propagation updates—so as to enable computation of so-called stable forwarding tables that are guaranteed to be loop-free while relying on transient routing heuristics in the forwarding plane to re-route packets encountering blackholes. Consensus routing can be viewed as logically centralizing the snapshottting of network state but continuing to rely on distributed BGP for computation of the stable forwarding tables; the snapshots simply slow down FIB changes making routers jump from one set of consistent FIBs to another.

The above body of prior work informs the design of CIRCA, a logically centralized, scalable, and fault-tolerant architecture to offload interdomain route computation as-is to the cloud without requiring ASes to reveal any additional proprietary policy information while ensuring predictably quick convergence delays, a combination of goals that to our knowledge has not been achieved before.
3 CIRCA design and implementation

CIRCA is driven by the following key design goals.

1. **Limited disclosure**: CIRCA should not require ASes to disclose proprietary policies or topology to any entity.

2. **Quick convergence**: CIRCA should ensure quick convergence unlike BGP’s high tail latencies (§3.2).

3. **High availability**: CIRCA should ensure high availability despite component or network failures (§3.4.2).

4. **Route computation equivalence**: CIRCA’s computed routes should match those computed by BGP or any desired decentralized and safe routing protocol (§3.4.1).

5. **BGP interoperability**: CIRCA should gracefully co-exist with BGP and be incrementally deployable (§3.5).

3.1 Design overview

BGP route computation, even given centralized global topology and policy information, is a computationally hard problem [13] unless restrictive assumptions are made, e.g., under Gao-Rexford (GR) conditions, the complexity is linear in the size of the network. To our knowledge, in the absence of more restrictive assumptions than safety of routing policies, previously proposed approaches for BGP route computation are not qualitatively more efficient than simulating BGP’s path exploration process, i.e., some sequence of receive/import/adopt/export activations until convergence. Even logically centralized approaches based on SMPC [17], that in addition to GR constraints restrict policies to be next-hop-based and routes to be uniquely determined by the source-destination pair, compute BGP routes by simulating BGP’s path exploration. Algebraic formulations of the problem can potentially compute BGP routing outcomes more efficiently than asynchronously simulating path exploration, e.g., via a generalized Dijkstra-like algorithm [46] or iterative matrix methods that synchronously simulate generalized Bellman-Ford’s [45, 46] path exploration, but only under restrictive assumptions that BGP in practice is not known to satisfy, e.g., non-neighbor-based policies like “prefer customer but disprefer routes containing AS X” are neither left-distributive nor monotonic (also known as strictly increasing [8]), each of which is known to be a sufficient condition for computing destination-based forwarding routes efficiently.

CIRCA’s high-level design based on virtual routers replaying BGP in the cloud is naturally dictated by two premises: (1) we need to limit disclosure of proprietary policy information to no more than what is shared between ASes in today’s BGP (also referred to as ground-BGP); (2) even with centrally disclosed policy information, without more restrictive assumptions than just safety, we don’t know of a qualitatively more efficient way to compute BGP’s routing outcomes other than to simulate its path exploration. Accordingly, CIRCA maps virtual router incarnates (or avatars) in the cloud to each ground router. In what follows, we first describe unreplicated CIRCA (deferring CIRCA’s replication mechanisms for ensuring high availability amidst failures to §3.4.2) wherein each ground router is one-one mapped to a cloud avatar and the cloud avatars execute a protocol derived from the underlying distributed ground routing protocol.

CIRCA operates incrementally in response to root cause events, i.e., external events such as a node or link up/down event, link cost change, or an operator-induced policy or configuration change. CIRCA operates in three phases—(1) detection and cloudward dispatch of a root event by a ground router; (2) route computation by cloud routers; (3) adoption of forwarding outcomes dispatched by cloud routers groundward—a high-level design also shared by several prior works [2, 4, 17, 26, 41].

3.1.1 CIRCA north-south protocol

The CIRCA north-south protocol consists of three phases.

**Phase I**: Upon detecting a root event, a ground router \(E\):

1. Assigns a unique label \(E = [\text{seqn}, R]\) to the event, where \(\text{seqn}\) is a sequence number incremented by exactly one for each locally detected root cause event;

2. Appends \(\langle E, \text{iface}, \text{etype}, \text{eval} \rangle\) to a local persistent log, where \(\text{iface}\) identifies the interface affected by the event and \(\text{etype}\) and \(\text{eval}\) are respectively the event type and value, e.g., \(\text{etype}\) may denote a link cost change and \(\text{eval}\) the value of the new link cost;

3. Sends the ground root cause (GRC) message \(\langle \text{GRC}, E, \text{iface}, \text{etype}, \text{eval} \rangle\) to its cloud avatar(s) \(v(R)\).

**Phase II**: CIRCA cloud routers then take these steps:

1. Upon receiving \(\langle \text{GRC}, E = [\text{seqn}, R], \text{iface}, \text{etype}, \text{eval} \rangle\), the cloud avatar \(v(R)\) initiates a distributed route computation algorithm (in §3.3) for that event after it has sequentially completed the execution for all root events \(\langle k, R \rangle\) for \(k < \text{seqn}\);

2. When a cloud router \(v(Q)\) impacted by \(E\) eventually receives the termination notification (as guaranteed by the liveness property in §3.3.2) for \(E\), it
   (a) appends the FIB entries updated as a result of \(E\) to a local persistent log;
   (b) dispatches the FIB entries to ground incarnates \(Q\);

**Phase III**: A ground router \(Q\) receiving FIB entries for \(E\):

1. Appends the FIB entries to a local persistent log;

2. Applies the FIB entries for \(E = [\text{seqn}, R]\) if it has applied all FIB entries for root events \(\langle k, R \rangle\) for \(k < \text{seqn}\);

The high-level protocol above can be optimized as follows. **Garbage collection**: A ground router \(Q\) informs its cloud avatar \(v(Q)\) to garbage collect its log entries for any root cause event \(E\) for which \(Q\) has already applied (in step III.2 above) the updated FIB entries. The ground router only persists \(O(1)\)
An important reason interdomain routing suffers from long convergence delays is the presence of MRAI (min route advertisement interval) timers [3, 28, 31, 36, 43, 49]. A nontrivial body of prior work suggests that MRAI timers are a necessary evil to tame BGP’s message complexity. Early work by Labovitz et al. [28] suggests that without such timers, BGP’s message complexity may be superexponential (or $O(n!)$). Although the gadgets exemplifying the superexponential message complexity in that work are somewhat contrived in that their policy configuration does not satisfy Gao-Rexford (GR) conditions that are believed to hold commonly in practice, subsequent work [9] has shown that BGP’s message worst-case complexity can be exponential even in GR topologies.

Our position is that MRAI timers even in ground-BGP today are overly conservative and are set to high values (e.g., 30s is a commonly recommended default value [31, 50, 51]) in part because of prohibitive message complexity and overall convergence delay is poorly understood. Classical work on this topic [12] suggests that convergence delay exhibits a nonmonotonic relationship to MRAI timers, i.e., there is a minima or a sweet spot setting for the timer below which operators risk worsening convergence delay because of prohibitive message complexity and above which the timers themselves are too conservative exacerbating delay. There isn’t universal agreement on what value of the timer is optimal; some have suggested that the common default of 30s is too high [12, 19, 31] while others have noted the risks of heterogeneous timer values further exacerbating message complexity [9], so conventional operational wisdom has been to err on the conservative side.

### 3.2 MRAI: Unnecessary evil in the cloud

An important reason interdomain routing suffers from long convergence delays is the presence of MRAI (min route advertisement interval) timers [3, 28, 31, 36, 43, 49]. A nontrivial body of prior work suggests that MRAI timers are a necessary evil to tame BGP’s message complexity. Early work by Labovitz et al. [28] suggests that without such timers, BGP’s message complexity may be superexponential (or $O(n!)$). Although the gadgets exemplifying the superexponential message complexity in that work are somewhat contrived in that their policy configuration does not satisfy Gao-Rexford (GR) conditions that are believed to hold commonly in practice, subsequent work [9] has shown that BGP’s message worst-case complexity can be exponential even in GR topologies.

Our position is that MRAI timers even in ground-BGP today are overly conservative and are set to high values (e.g., 30s is a commonly recommended default value [31, 50, 51]) in part because of the relationship between MRAI timers, message complexity, and overall convergence delay is poorly understood. Classical work on this topic [12] suggests that convergence delay exhibits a nonmonotonic relationship to MRAI timers, i.e., there is a minima or a sweet spot setting for the timer below which operators risk worsening convergence delay because of prohibitive message complexity and above which the timers themselves are too conservative exacerbating delay. There isn’t universal agreement on what value of the timer is optimal; some have suggested that the common default of 30s is too high [12, 19, 31] while others have noted the risks of heterogeneous timer values further exacerbating message complexity [9], so conventional operational wisdom has been to err on the conservative side.

### 3.2.1 Why the cloud is different

There are three critical differences between the cloud and ground-BGP with implications for MRAI and other timers.

1. **Control bandwidth**: The CIRCA cloud can be easily provisioned with 2-3 orders of magnitude more control bandwidth (e.g., tens of Gbps) compared to typical control traffic provisioning in ground-BGP today.

2. **Propagation delay**: The propagation delay diameter of the CIRCA cloud is 2-3 orders of magnitude less than the hundreds of milliseconds in ground-BGP today.

3. **Control-data isolation**: Interference between control and data is a non-concern in the CIRCA cloud, so it can afford to be much more wasteful than ground-BGP.

Accordingly, we pick an aggressive point in the design space for CIRCA, namely, no MRAI timers at all! The justification for this design choice is as follows. First, MRAI timers, when they help, intuitively work by limiting spurious path exploration by making each node wait long enough to hear from its neighbors about their reaction to earlier routing messages. However, these very timers can make nodes unnecessarily wait for several MRAI rounds even when there is no possibility of excessive spurious path exploration. Worse, some (admittedly contrived) choices of heterogeneous timer values even in GR settings can actually cause exponential message complexity [9] even though overall convergence delay in GR settings is determined by the speed of information propagation along the customer-provider chain and back. Eliminating MRAI timers altogether propagates information as quickly as possible while naturally slowing down node reaction times because of message queuing delays in the rare cases that message processing is indeed a bottleneck.

Second, message processing delays in commodity routers as well as processing delays in well provisioned cloud environments are an order of magnitude lower than the numbers used in prior simulation-based work [12, 43]. Third, we hypothesize that reasonable values of MRAI timer values when useful are positively correlated with propagation delays, i.e., the higher the propagation delays, the higher the timer values needed, all else being equal. This hypothesis is consistent with (but not strictly implied by) the simple model in [37].

Our prototype-driven experiments in §4.1 validate the hypotheses above showing both that MRAI timers are unnec-
cessary in reasonably provisioned cloud settings and that our results do not qualitatively contradict the nonmonotonic behavior reported in previous works when compute provisioning is artificially limited to unrealistically low levels.

3.3 Distributed convergence detection

The description thus far has side-stepped a critical question: How do CIRCA cloud routers replaying a distributed protocol like BGP know when the processing of a root cause event has converged? A convergence detector is necessary for a cloud router to initiate transmission of the corresponding updated FIB entries to its ground incarnate. The distributed BGP protocol as executed by Internet routers today does not have any such indicator of convergence (other than, trivially, the absence of any routing updates for a sufficiently long time).

To appreciate why detecting convergence in CIRCA is not easy, consider the strawman strategy of each virtual router periodically, say once every \( \tau \) seconds, reporting the most recent message it processed for a root cause event \( e \) to a centralized monitor that declares convergence when no router has processed any message for \( e \) in its most recent reporting interval. For \( \tau = 5 \)s and a total of say \( 10^6 \) routers, this reporting traffic alone for each root event is 200K pkts/s. A hierarchical convergence detector that employs a per-AS detector with 50K ASes will still incur a reporting traffic of 10K pkts/s per root event, which is prohibitively expensive especially given that many root events may have only a localized impact affecting the forwarding behavior of a small number of ASes if at all. More importantly, a centralized monitor will incur a convergence delay of at least \( \tau \) seconds per root event, not to mention poor fault tolerance. Finally, a centralized monitor by design observes more information than can be observed or easily inferred by any AS in BGP today, thwarting CIRCA’s limited disclosure goal (§3.0).

We present a completely distributed convergence detector that is provably safe, i.e., no false positives, and live, i.e., it will detect convergence provably within at most \( 3 \times \) the actual convergence delay—and in practice within a much smaller factor (§4.2)—and the number of BGP messages as observed by a (theoretical) global monitor. The distributed convergence detection algorithm works in three phases as explained below. For ease of exposition, we first assume failure-free execution (deferring fault tolerance to §3.4). The algorithm is bootstrapped with a set of path updates generated by simulating the root event at the “root” cloud router receiving the corresponding ground root cause message.

Figure 2 illustrates the three phases of the convergence detection algorithm. In the first exploration phase, for each root event \( e \) processed by a virtual router, the router maintains state about the event by adding it to a set of unprocessed events. When a virtual router \( R \) processes a BGP message \( m \) related to \( e \) and it induces no change to \( R \)’s FIB (and consequently no new announcements to its neighbors), we say that message \( m \) “fizzled” at \( R \). For each fizzled message, \( R \) back-propagates aizzle indicator to its peer that previously sent it to thereby initiating the second back-propagation phase. Each router locally keeps track of each message pertaining to \( e \) that it sends to each neighbor and waits for the corresponding fizzle acknowledgement to be back-propagated. When the router that initiated the root cause event has received fizzle acknowledgements for all messages it originated, it determines \( e \) to have converged, which initiates the third dissemination phase wherein, starting with the root router, each router forwards the convergence indicator along each link previously traversed in the exploration phase, at which point it drops all state related to \( e \) for convergence detection.

3.3.1 Formal event-action protocol

In order to formally reason about the safety and liveness properties of the distributed convergence detector, we codify it in event-action format in Algorithm 1. The key notation is as shown in Table 1. Where there is little room for confusion, we drop the argument \( \mathit{R} \) implicitly fixing it to \( \mathit{self} \) at a router.

Algorithm 1 shows how a router \( \mathit{R} \) handles three distinct events: (1) receipt of a root cause message \( \langle \mathit{GRC}, \mathit{E} \ldots \rangle \) from its ground incarnate \( \mathit{v}(\mathit{R}) \) that initiates path exploration; (2) receipt of a CBGP message from a peer cloud router; and (3) receipt of a FIZZLE message from a peer cloud router.

The first event handler processes the received root cause message by “simulating” the corresponding ground event, which produces a set of resulting update messages announcing or withdrawing a set of paths to affected prefixes, a set denoted as \( \mathit{paths}(E) \). If the root event \( E \) does not change \( \mathit{R} \)’s FIB, then routing has trivially converged (line 7). Else, for each changed entry in \( \mathit{R} \)’s FIB causing a new update subject to its export policy, it creates a unique timestamp as a two-tuple consisting of a strictly increasing logical clock returned by \( \mathit{now}() \) and the router’s identity. (This logical clock does not need to reflect the happens-before relationship between events like Lamport clocks for reasons that should be clear from the formal proofs in §A.) The router stores each resulting update to its peers in a map \( \mathit{sent}[\mathit{E}] \) and remembers the cause of each sent update...
as the original GRG message in a cause map indexed by the
timestamp ts of the sent update. By definition of now(), each
sent update has a unique timestamp.

The second event handler, the common-case action invoked
at a cloud router beyond the root router, is similar to the first
but with two important differences. First, if a received update
(CBGP, E, p, asPath, ts) from a peer does not change its FIB,
it responds with the corresponding (FIZZLE, E, ts). Second,
if it does change its FIB, it remembers the cause of each
resulting export-policy-filtered update as the incoming update.

The third event handler purges entries from the send[E] map
upon receipt of the corresponding fizzle messages (removing
the message with timestamp ts from the set send[E][ts] in line
31). If the send set caused by an update is emptied at a non-
root router, it back-propagates a fizzle to the peer that sent the
incoming causal update (line 35). When an incoming fizzle
message empties the send[E] map at the root router (line 33),
it declares the distributed processing of event E as converged,
and initiates the third dissemination phase (deferred to §A).

We need to formally prove the correctness property that
when a root router thinks routing has converged, that is indeed
the case; and the liveness property that the above protocol
will eventually terminate under realistic conditions.

3.3.2 Safety, liveness, and efficiency

The formal proofs of all claims herein are deferred to §A.

Theorem 3.1. Safety: If converged(E) is true at the root
cloud router that received the GRG, then no further CBGP
message for E will be received by any cloud router.

Theorem 3.2. Liveness: Algorithm 1 eventually terminates,
i.e., converged(E) is set at the root router, if CBGP is safe1 and
all cloud routers are available for sufficiently long.

The safety and liveness proofs rely on a construction called
the directed message graph produced by any execution in-
stance of Algorithm 1, as illustrated in Figure 3, wherein each
vertex corresponds to a message—either the original GRG or a
CBGP message—and there is a directed edge from a message
m1 to another message m2 if m1 caused m2, i.e., m2 was sent
in lines 14 or 26 in response to the receipt of m1 in lines 1 or
16 respectively. We say that m1 → m2 (read as m1 happened
before m2) if there exists a directed path from m1 to m2 in
the graph. It is straightforward to show (Lemma A.1) that
the directed message graph is a directed tree, i.e., it is acyclic
each vertex has exactly one incoming edge. The proof
of safety relies on showing that if m1 → m2, then m1 fizzes
only after m2 has previously fizzled (Lemma A.3) and when
converged(E) is true at the root router, no CBGP messages for
E are under propagation anywhere (Lemma A.4). The proof
of liveness relies on showing that, with safe CBGP policies,
every CBGP message eventually receives a matching FIZZLE
(Lemma A.5) as well as on Lemma A.3 and on the assumption

1CBGP is safe” means that it is guaranteed to converge to a stable route
configuration [13] and is unrelated to safety in the theorem just above.
Theorem 3.3. **Efficiency**: The root router (Any router) in Algorithm 1 detects convergence within at most $2\Delta (3\Delta)$ time and $2M (3M)$ messages where $\Delta$ and $M$ are respectively the actual convergence delay and number of messages incurred by the distributed route computation in the cloud.

Although a $3 \times \Delta$ delay overhead may seem high and it may cursorily appear possible to reduce that overhead with simple optimizations, our attempts at doing so while preserving provable safety and liveness have been elusive. Fortunately, our experiments (§4.1) show that (1) convergence delays in the cloud are significantly smaller than those in the ground, so a $3 \times \Delta$ overhead in the cloud is still a big net win; and (2) the overhead is much smaller than $3 \times \Delta$ because messages in the first exploration phase can contain a large number (even thousands) of prefixes, but messages in the other two phases are like small acknowledgments.

### 3.3.3 Concurrent event processing

The discussion above implicitly focused on sequentially processing one root event at a time. However, we can not afford the luxury of a centralized monitor or other mechanisms to ensure sequential processing of root events for the same reason that convergence detection had to be distributed in the first place. With multiple concurrent root events being processed by different cloud routers, Algorithm 1 in conjunction with the high-level end-to-end protocol in §3.1.1 has a problem: the FIBs dispatched by a cloud router in step II.2.b may be inconsistent as they may reflect the incomplete processing of one or more root events being concurrently processed in the system, which in turn could result in transient loops or blackholes in the data plane at ground routers (as can happen in ground-BGP today even with just a single root event). However, the safety and liveness properties above as well as Route Computation Equivalence as formalized in the next subsection still hold, so our CIRCA implementation simply processes concurrent root cause events in parallel. A more detailed discussion of the pros and cons of concurrent event processing while preserving route consistency in the ground data plane is deferred to a technical report [38].

### 3.4 Route Computation Equivalence despite link or router failures

Informally, this section shows that any decentralized policy routing (or “ground”) protocol, including but not necessarily limited to BGP, satisfying a naturally desirable consistency property (ECC, as formalized below) can offload its route computation to CIRCA with the guarantee that the CIRCA-equipped system will eventually achieve the same routing outcomes as the unmodified ground protocol despite failures, a property referred to as Route Computation Equivalence. Unlike the safety and liveness properties shown for CIRCA’s cloud control plane, the results in this section subsume the data plane or end-to-end forwarding behavior on the ground.

**Network state model.** We model the ground network as a state machine with root cause events effecting state transitions. The state of the network encompasses the (up/down) state of all links and routers as well as any configuration information (link costs, operator-specified policies at a router, etc.) that potentially impacts routing outcomes. Let $S_0$ denote the initial state of a network and $[e_1, \ldots, e_k]$ denote a sequence of root cause events such that each event $e_i$ transitions the network from state $S_{i-1}$ to state $S_i$. The state of the network after an event or a sequence of events is denoted using the operator $''|$ as in $S_1 = S_0|e_1$ or $S_k = S_0|[e_1, \ldots, e_k]$.

**Definition 1.** **Eventually Consistent Convergence** (ECC): If no root cause events occur for a sufficiently long period, forwarding behavior should converge to reflect the state of the network just after the most recent root cause event.

We posit essentially consistent convergence as defined above as an intrinsically desirable property for any practical routing protocol. ECC as defined is rather weak because “eventual” allows routes to be inconsistent with global network state for arbitrarily long, but BGP today does satisfy this property provided routing policies are safe (or conver-
gent) and every event is eventually detected and acted upon by incident routers immediately impacted by it.

3.4.1 Unreplicated CIRCA ensures RCE

We next formally define Route Computation Equivalence.

Let $D(S)$ represent any distributed route computation function that, given network state $S$, returns a set of possible routing outcomes, i.e., a set \{GFIB$_1$, GFIB$_2$, \ldots\} wherein each element represents global forwarding behavior as captured by the union of the FIBs of all routers in the network. The reason $D(S)$ is a set of size greater than one is to incorporate non-determinism as BGP even with safe policies in general can have multiple stable routing configurations, e.g., the DISAGREE gadget [13] converges to one of two possible stable routing outcomes depending on node reaction times.

Let $GFIB(t)$ denote the forwarding routes adopted by ground routers at time $t$. We would like to show that if a distributed route computation process satisfies ECC, CIRCA preserves those ECC routing outcomes. Formally,

**Definition 2. ROUTE COMPUTATION EQUIVALENCE (RCE):**

Given an initial network state $S_0$ and a finite sequence of root case events $[e_1, \ldots, e_n]$, a cloud-assisted routing protocol is said to preserve equivalence with respect to a distributed route computation function $D(S)$ if it ensures that $\lim_{t \to \infty} GFIB(t) = D(S_n)$ where $S_n = S_0 \parallel [e_1, \ldots, e_n]$.

Next, we show that a single-datacenter (or unreplicated) pure CIRCA deployment ensures RCE despite intermittent failures (proof deferred to Appendix B), where pure means all ground routers have been upgraded to rely only on CIRCA.

**Theorem 3.4.** If for a sufficiently long period—(i) all ground routers can reach a CIRCA cloud replica and vice versa; and (ii) all cloud routers are available and can communicate in a timely manner—a pure CIRCA system ensures Route Computation Equivalence with any distributed route computation function that satisfies Eventually Consistent Convergence.

3.4.2 Replicated CIRCA ensures RCE

To see why Theorem 3.4 holds even in a replicated CIRCA deployment without any replica coordination protocol, we simply observe that each CIRCA replica independently ensures RCE, i.e., the FIBs it computes reflect the most recent state of the network provided it eventually receives all root cause event reports (in any order). If each ground router is responsible for relaying each root event to all of its cloud avatars, no replica coordination protocol between CIRCA replica sites is necessary. CIRCA cloud routers may optionally relay root events to its siblings on other replica sites as an optimization, but this is not necessary for ensuring Route Computation Equivalence. An analogous observation, namely that no sophisticated replica coordination protocol is needed for safety, has been long known for a single-domain route computation service (e.g., RCP [4]), but not for interdomain routing. Furthermore, the technical reasons why they hold in the CIRCA architecture are processing root cause events consistently are very different from RCP that relied on an assumption of consistent views of the ground network despite partitions across replicated route servers.

**Overhead of faults.** CIRCA’s simple design maintains RCE despite arbitrary failure patterns but failures, specifically of cloud routers or link failures inducing ground-cloud unreachability, have two costs: (1) growing log of unprocessed root events at ground routers, and (2) transient forwarding loops or blackholes. As detailed in the techreport [38], the former is bounded by the size of the total configuration state at a router and the latter can be alleviated (but not eliminated) by replicating CIRCA datacenters in a pure CIRCA deployment or by relying on BGP co-existence mechanisms below.

Conveniently, ground router failures are a non-issue because of the fate sharing property that it is the only router stalled by its failure; neighboring ground routers will detect and process its failure as a normal root cause event.

3.5 Co-existence & incremental deployability

**Co-existence** refers to the ability of ground routers to leverage the CIRCA cloud while continuing to rely on ground-BGP as a fallback or an optimization under the (possibly impractical) assumption that all ground routers have been upgraded to be CIRCA-capable. Incremental deployability refers to the ability to gainfully deploy CIRCA despite upgrading only a subset of ground routers to be CIRCA-capable. (As defined, a deployment can not be both co-existent and incremental.)

3.5.1 Co-existence with ground-BGP

CIRCA’s support for co-existence enables ground routers to get the best of both worlds, i.e., adopt forwarding outcomes from whichever plane—ground or cloud—converges earlier, thereby also implicitly relying on ground-BGP alone as a fallback when the cloud is unreachable. To this end, ground-BGP needs to be extended so as to tag each message with the corresponding root cause event label, and ground-BGP routers need to relay the tag in any incoming message by inserting it into any outgoing messages caused by it.

A ground router $R$ uses the root event label $E = [seqn, R]$ in a ground-BGP message as follows. If $R$ has already received $\Delta FIBEntries(E)$ for $E$ from the cloud, it rejects any further ground-BGP messages tagged with $E$ from impacting its FIB, otherwise it continues processing ground-BGP messages as usual through its decision engine with one difference: it marks any changes to its FIB as a result of $E$ as such and keeps a copy of the original entry for the corresponding prefix, which essentially allows $R$ to undo the effect of any ground-BGP messages tagged with $E$. When $R$ eventually receives $\Delta FIBEntries(E)$ from the cloud, irrespective of whether or not ground-BGP has already converged at $R$ (necessarily unbeknownst to it), it installs $\Delta FIBEntries(E)$ anyway undoing the effect of any ground-BGP messages pertaining to $E$. 
Co-existence further requires ground routers to maintain a route information base (RIB), or the set of available routes, unlike universal CIRCA (wherein every router is CIRCA-capable) that only required ground routers to maintain a FIB (or the set of adopted routes). Maintaining a RIB plus a FIB is no different from ground-BGP today, and requires ground routers to continue processing messages tagged with \( E \) so as to update its RIB (but not its FIB) even after it has received the corresponding \( AFIBEntries(E) \) from its cloud incarnate. Maintaining a RIB in co-existent CIRCA also enables ground routers to employ fast reroute or backup routing options during the ground or cloud convergence period.

### 3.5.2 Incremental deployability

Incremental deployment means that some ground routers may be completely CIRCA-unaware, so such legacy routers can not even parse BGP messages with root cause labels. In such scenarios, the benefits of CIRCA would be limited to root cause events that originate within and whose exploration fizzes entirely within a contiguous island of routers all of which have been upgraded to be CIRCA-capable. For events that entirely fizzle within the contiguous upgraded island, the protocol is identical to the co-existent scenario (§3.5.1). If an event spills out of this island, it is recognized as such by any CIRCA cloud router at the island’s boundary. When a cloud router detects a spill-over, it immediately aborts the exploration by back-propagating an ABORT message (instead of a FIZZLE), enabling the root router to conclude that the CIRCA cloud can not compute the outcome of that event after all.

CIRCA needs an additional mechanism in order to preserve RCE in incremental deployment scenarios, in particular, to ensure that cloud routers do not diverge from their ground counterparts. To this end, each boundary router in the CIRCA cloud, i.e., any router whose ground incarnation has at least one neighbor that has not been upgraded to be CIRCA-capable, establishes receive-only ground-BGP sessions with those (ground) routers, wherein a receive-only session is one that only receives, but never sends, any announcements. Note that, notwithstanding this additional peering requirement, because of the non-deterministic nature of ground-BGP, it is possible for some events to (not) fizzle within the upgraded island in the cloud even though they might have not fizzled (fizzled) within the corresponding island on the ground. §B formally shows why RCE is nevertheless preserved under incremental CIRCA deployment scenarios.

### 3.6 CIRCA implementation

We implemented a Quagga-based prototype of CIRCA including the complete north-south protocol and the distributed convergence detection algorithm as described in roughly 2,430 lines of code. We have not implemented co-existence or incremental deployability mechanisms (§3.5) but evaluate their benefits via simulations based on real AS topologies.

![Figure 4: Header and body of a CIRCA BGP packet](image-url)
(150 physical machines with 1200 cores), we extract contiguous subgraphs of the AS topology of varying sizes from 20 to a maximum of 1200 ASes. For extracting a subgraph of \(n\) ASes from the CAIDA data set, we first randomly pick an AS and perform a breadth-first search until the number of selected nodes reaches \(n\). We increase the network size by only adding new nodes to the smaller subgraphs. The average degree of nodes and the number of prefixes belonging to each AS in our extracted topologies are shown in Table 2.

<table>
<thead>
<tr>
<th>Number of routers in topology</th>
<th>20</th>
<th>60</th>
<th>180</th>
<th>540</th>
<th>1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average degree of routers</td>
<td>4</td>
<td>8</td>
<td>13</td>
<td>19</td>
<td>29</td>
</tr>
<tr>
<td>Average number of prefixes</td>
<td>225</td>
<td>296</td>
<td>180</td>
<td>155</td>
<td>106</td>
</tr>
<tr>
<td>All unique prefixes</td>
<td>4.5k</td>
<td>17.8k</td>
<td>32k</td>
<td>82.7k</td>
<td>125.3k</td>
</tr>
</tbody>
</table>

Table 2: Properties of used topologies.

Our experiments retain Quagga’s so-called “burst” MRAI timer implementation where all updates to a given peer are held until the timer expires, at which time all queued updates are announced. Quagga applies the MRAI timer only on updates, not on withdrawals, a behavior we retain. The published official BGP4 protocol specification RFC 4271 [39] suggests that route withdrawal messages are also to be limited by the MRAI timer; a change from earlier version (RFC 4098) where withdrawals could be sent immediately [40].

We emulate four different root cause events in our experiment; node up, node down, link up and link down. For each network size and MRAI value, we first assign the routers in the network to physical machines randomly and set the same MRAI timer on all routers and then wait for a long enough duration until routes to all prefixes at all routers stabilize. At this point, we randomly pick a router in the topology and trigger a random root cause event, and repeat this process 30 times for each topology size and MRAI value. We log each BGP message and post-process log files from all routers at the end of each run to compute the convergence delay.

Figure 5 shows the average convergence delay of 30 simulated root cause events using different MRAI values. In computing the convergence delay of each root event, we do not count the time it takes for the root router to detect the event because this delay (≈ one to few seconds) will be incurred by ground routers but not CIRCA cloud routers. We consider the time when the last immediately impacted incident router detects the event as the beginning of the convergence period.

A couple observations are noteworthy. First, the absolute values of the convergence delay with zero MRAI timers are small. Convergence delay increases as expected with the topology size and number of affected prefixes of root router and the highest (average) value observed is 2.3 seconds with 1200 ASes. We conduct a smaller-scale experiment with multiple routers per AS using realistic intradomain topologies (deferred to a techreport [38]). Unsurprisingly, the qualitative trend of increasing convergence delays with increasing iBGP

Figure 5: Average convergence delay of BGP across all root events with different MRAI timers on Emulab.

MRAI values persists. Second, convergence delay monotonically increases with the MRAI timer value and a zero timer yields the lowest convergence delay. This observation is in contrast to an established body of prior research [9, 12, 28, 43] suggesting a non-monotonic relationship between MRAI timer values and convergence delays and the existence of a minima for convergence delay at an MRAI value greater than zero, and that lower or zero MRAI values can significantly exacerbate convergence delays.

There is no contradiction however with prior findings. Previous work has been largely based on simulations and toy topologies (e.g., [12, 43]) and assumed unrealistically high values of message processing delays, e.g., random delay from zero to 1 second for each message in [12] and a uniformly distributed delay between 1 and 30 milliseconds for message processing [43]. There is no fundamental reason for message processing delays to be that high on modern processors and we find that they are at best tens of microseconds in Quagga (after we systematically disabled unnecessary timers).

4.1.2 Reconciling prior findings

As a sanity check, to verify that with artificially inflated message processing delays, we can reproduce the nonmonotonicity observed in prior work, we reduce the number of physical machines in our Emulab setup from 150 to 50 machines and re-run the experiment with the largest topology size (1200). With this setup, each 8-core machine now has almost 24 Quagga instances running on it compared to roughly eighth Quagga instances per machine (or one Quagga instance per core) in the earlier setup. Quagga is single-threaded software, so a Quagga instance can not leverage other cores on a physical machine even if they are idle.

Figure 6 shows that the nonmonotonic behavior is reproducible with more compute stress. However, it is important to clarify that this behavior is not solely because of the modest 3× decrease in resource provisioning. Because Quagga is a single-threaded software, an MRAI timer of 0 causes it to spin in a busy wait consuming 100% CPU utilization on each core even with one core per Quagga instance, a problem that exacerbates with 3-4 Quagga instances per core. Without this implementation artifact of wasteful busy waiting, we were unable to reproduce the nonmonotonic behavior observed in
prior work. Our observations therefore suggest that, with modern processing fabrics in routers, it may be worth revisiting whether conservatively set high MRAI values are appropriate even in ground-BGP today.

4.1.3 Extrapolating our findings to 60K ASes

Can even larger topologies going all the way up to 60K ASes result in prohibitively high message complexity exacerbating convergence delays? We think it unlikely for several reasons.

First, most root events are unlikely to impact a very large number of ASes unless a prefix is rendered altogether unreachable [36]. Second, even if an event impacts all ASes, with realistic Gao-Rexford policies, although pathological cases of exponential message complexity are theoretically possible, they require contrived topologies and unrealistic assumptions about relative speeds of routers to manifest in practice [22]. Even so, it may be possible to manage resource more effectively, e.g., we show in an experiment (deferred to a technical report [38]) that there is room to improve the core utilization by 30× in our testbed with multi-threading and careful mapping of virtual routers to cores. Third, even if many routers get affected by some root cause events, it is unlikely that the FIB entries (as opposed to just RIB entries) on most of the routers will change. Fourth, if message complexity does become prohibitive, with ultra-low LAN propagation delays in the CIRCA cloud, queued messages can be batch-processed (with no intervening exports) so as to provide a self-clocking mechanism to slow down announcements similar in spirit to MRAI timers but without relying on conservative, static values. This idea is similar to the proposed adaptive or dynamic MRAI values wherein the MRAI value is increased if there is more "unfinished work" in the queue and decreased if not [32, 43].

4.2 End-to-end convergence delay

In this section, we evaluate end-to-end convergence delay including the overhead of distributed convergence detection and north-south communication delays for a root cause event.

**Figure 6**: Non-monotonic trend of convergence delay vs. MRAI timer value with the 1200-AS topology.

**Figure 7**: Difference between actual convergence delay in cloud, convergence detection delay using our algorithm, and ground convergence delay.

### 4.2.1 Convergence detection overhead

First, we evaluate the overhead introduced by the distributed convergence detection algorithm (Algorithm 1) to detect convergence after the protocol actually converges in cloud. As before, we estimate the ground-truth convergence delay by processing router logs and compare it to the time reported by our algorithm and repeat each root event injection 20 times.

We conduct this experiment with 20, 60, 180, and 540 ASes respectively in the ground and cloud setups with 20 randomly simulated root cause events in the ground setup in isolation. On ground, we set MRAI 2 and 4 seconds. Each ground router has a connection with its avatar in the cloud and both advertise the same list of prefixes in the network. We consider three different root cause event types in the ground network; node down, link up and link down. We have a single router AS setup for this experiment. The average degree of nodes in our topology and the number of prefixes belonging to each AS are shown in Table 2.

Figure 7 shows the average of the detected convergence time and the ground-truth convergence delay in cloud and also on ground for all root cause events for different topology sizes. Per design, we might expect the root router to take roughly 2× the time to detect and for the last router to take up to 3× time, but the observed overhead is much smaller. The reason for this fortuitous discrepancy is that messages in the first exploration phase usually contain a large number (even thousands) of prefixes, which increases processing and transmission times in that phase, but not in the other two phases.

### 4.2.2 End-to-end delay in getting new FIB entries

In this experiment, we evaluate the end-to-end delay for obtaining new FIB entries from the cloud.

We assume the ground routers in an island with different sizes have been upgraded to CIRCA. We run the ground instance of the routers and their avatars in a LAN (our Emulab CIRCA testbed). While we can find the convergence detection time of each router using our implemented CIRCA system, for measuring ground and cloud communication delay, we can not use the delay between ground routers and their avatars in the LAN as real Internet delay. We can estimate the
delay from Internet routers to their avatars by pinging routers in target ASes at the ground from their avatars at the cloud. The prefixes of each AS on the Internet are identifiable from the Routeviews data set [42]. For five original prefixes of our target ASes, we generate an IP address for each prefix by adding 100 to the end of the prefix, e.g., 10.10.8.100 for the prefix 10.10.8.0/24. We ping target IP addresses three times and get the average ping delay from that AS to our data center. For the roughly 20% of IP addresses for which we do not get any result, we assume 100 ms as the one-way delay.

We conduct this experiment for varying topology sizes with 20 randomly simulated root cause events on the ground in isolation. We consider 20, 60, 180, and 540 ASes on the ground and in the cloud.

Figure 8 shows the end-to-end delay of getting the first entry in the FIB table to ground routers in an island with different sizes after simulating the root cause event on the ground. The end-to-end delay is the sum of the delay in sending event data to the cloud (approximated crudely as the ping delay), detecting the stable state of the network by each router, and receiving the new FIB entries from the cloud (ping delay) across all root cause events and routers affected by our root cause events. In small topologies, most of the routers (80%) get the FIB entry across all root cause events in less than 400 ms. However, for our biggest topology size, 540 nodes, we have 1.5 seconds for around 80% of routers. As explained in section 4.1.3, most of the delay is because of the delay in the first phase of our convergence detection algorithm and could be further optimized.

### 4.3 Incremental deployment of CIRCA

We evaluate the fraction of root cause events that fizzle entirely within an island of upgraded single-router ASes chosen as a subset of the Internet’s AS topology. We sequentially simulate 20 randomly chosen root cause events originating within each upgraded island. We consider different approaches for selecting the upgraded routers: (1) tier-1 that first “upgrades” the contiguous ASes that do not have any provider and then their customers, and so on; (2) random that picks ASes in the network randomly (possibly in a non-contiguous manner); and (3) chain that picks a random contiguous chain of customer-provider ASes. We do not consider stub ASes in our experiment as candidate ASes for upgrading.

Figure 9 shows the fraction of root cause events that fizzle entirely within upgraded islands of different sizes with different approaches. For example, for covering 40% of all root cause events, we need to upgrade roughly 80% of routers with the chain or tier-1 approaches.

### 5 Discussion: Security, privacy, open issues

Our first stab at a practical interdomain-routing-as-a-service system leaves open questions that merit further discussion.

Any new system inevitably introduces new vulnerabilities. Our high-level goal in this work was to limit misbehavior to what is already possible in BGP’s control plane, and to limit information disclosure explicitly by design, however CIRCA itself introduces new security vulnerabilities as well as side-channel information leakage as discussed below.

**Side channel leakage.** First, CIRCA allows a rogue AS to exploit rapid convergence as a probing mechanism to infer policies of other ASes by willfully tampering its announcements and observing how others react, an attack also possible in “slow motion” in ground-BGP, and mechanisms similar to route flap damping may be necessary in the CIRCA cloud if such “fast-forwarded” probing attacks were to become a credible threat. FIZZLE messages expose a new side channel allowing an attacker to use the time between a Cbgp message and its corresponding FIZZLE to infer more information than is explicitly revealed by ground-BGP. Note that, given that every Cbgp is guaranteed to eventually elicit a corresponding FIZZLE, the information leaked by this side channel is limited to the convergence delay. Third, if a single provider owns the entire network infrastructure in the CIRCA cloud, it can monitor control traffic patterns (despite encryption) to derive more information than is realistically possible in ground-BGP today. We defer further analyses of and defenses against these and other information leakage attacks to future work.

**Security vulnerabilities.** CIRCA’s design allows BGP in the distributed cloud protocol to be drop-in replaced by S-BGP [23] (or other related security extensions like soBGP, BGPSec, etc.) while qualitatively preserving its convergence...
delay benefits as well as safety, liveness, and route computation equivalence guarantees provided the secured variant of BGP is guaranteed to produce routing outcomes equivalent to its unsecured version. CIRCA largely continues to be vulnerable to protocol manipulation attacks [47] against which cryptographic mechanisms as in S-BGP alone cannot protect, however known manipulation attacks based on abusing MRAI timers allowing an off-path adversary to potentially permanently disconnect good ASes despite the existence of good policy-compliant paths between them are ineffective in CIRCA because of its avoidance of MRAI timers.

CIRCA’s liveness guarantee (§3.3.2) relies on the safety of BGP policies, which potentially allows a rogue AS to change its policies (in a possibly self-defeating manner) simply to stall the CIRCA control plane. Although the rogue AS could mount this attack even in ground-BGP today, an ameliorating factor on the ground is that routers will continue to update their FIBs and forward traffic throughout the never-ending convergence period, potentially over loops or blackholes. In the CIRCA cloud in contrast, any root cause event—necessarily a policy change event (as opposed to a link/node down/up event)—that results in a safety violation will never converge, so ground routers will never receive the updated FIBs corresponding to that particular event. However, there are two alleviating factors in CIRCA. First, ground routers can fall back on ground BGP with CIRCA’s support for BGP co-existence for that root event. Second, other root events can proceed unaffected with CIRCA’s default “careless” concurrent event processing approach. These factors suggest that CIRCA-enabled BGP will be no worse than BGP in the presence of safety-violating policy attacks.

Non-convergent path exploration can be cleanly limited by augmenting the CIRCA cloud protocol with a TTL (time-to-live) mechanism that aborts (or force-fizzles) path exploration along any path in the directed message tree after a predefined maximum number of hops. A second design choice is to augment the cloud protocol with a TTL that upon expiration causes a cloud router to dispatch its current FIB to its ground avatar, reset the TTL to its maximum value, and resume the (never-ending) path exploration phase for that root event, a design that in effect induces ground routers to jump from one set of potentially inconsistent FIBs to another in a never-ending manner (similar to BGP). Such adaptations will not preserve RCE as defined because RCE is not well defined in the absence of ECC that will not hold under unsafe or non-convergent BGP policies, however the second design choice in practice comes close to emulating BGP behavior in non-convergent scenarios. We defer a more detailed design and analysis of CIRCA with unsafe BGP policies to future work.

**Honest-but-curious threat model.** CIRCA as described herein trusts each (replicated) cloud provider to provide and maintain the physical infrastructure in a non-curious manner, but its design can be extended to support a honest-but-curious cloud provider. One option is to employ emerging secure computing platforms [34] to prevent the entity controlling the physical machine from snooping on protected customer data within the machine, an approach that does however implicitly involve the manufacturer of the secure computing processor (e.g., Intel with SGX). To prevent a cloud provider controlling the OS on the machine from using the pattern of memory accesses from leaking information, further techniques such as Oblivious RAM [48] will be required. A quirker alternative is to organize each replicated CIRCA location similar in spirit to a global exchange point with a “bring-your-own-hardware” model in conjunction with physical security mechanisms, a design wherein the CIRCA cloud itself is federated obviating a trusted infrastructure provider.

**Security and robustness benefits.** Our hypothesis is that the long-term potential benefits of CIRCA are likely to outweigh the impact of new vulnerabilities it introduces. First, CIRCA provides a clean slate enabling early adopters to employ from the get go secure BGP mechanisms that have seen two decades of research and standardization work but little deployment in practice. Second, with willing AS participants, it is intrinsically easier to monitor select portions of control traffic in the CIRCA cloud compared to ground-BGP today in order to detect misbehavior such as protocol manipulation attacks. Third, CIRCA enables new opportunities such as augmenting the CIRCA cloud with mechanisms for “what-if” analysis allowing operators to ascertain whether an action will have the intended effect before performing that action.

### 6 Conclusions

In this paper, we presented the design, formal analysis, implementation, and prototype-driven evaluation of CIRCA, a logically centralized architecture for interdomain routing control. Although logical centralization of network control is a widely embraced direction in recent years in intradomain networks, attempts to extend that vision to Internet-wide interdomain routing have been limited to on-paper designs or small-scale evaluations. To our knowledge, this is the first work to present a full-fledged design and implementation of an interdomain routing control platform. Our underlying technical contributions include a novel distributed convergence detection algorithm; demonstrating the potential for significantly reducing BGP’s tail latencies; formally ensuring route computation equivalence with a broad class of BGP-like protocols, and designing mechanisms for enabling incremental deployment and coexistence with BGP.

### References


A Algorithm 1: Safety and liveness proofs

In this section, we formally prove the safety and liveness properties satisfied by Algorithm 1. The proofs rely on a construction called the directed message graph produced by any execution instance of Algorithm 1 and defined as follows.

Definition 3. DIRECTED MESSAGE GRAPH: The directed message graph of an instance of Algorithm 1 is a directed graph $G = (V, E)$ such that each vertex in $V$ denotes a sent message (either the original GR or a CBGP message) and $E$ has a directed edge from a message $m_1$ to another message $m_2$ if $m_1$ caused $m_2$, i.e., $m_2$ was sent in lines 14 or 26 respectively in response to the receipt of $m_1$ in lines 16 or 1.

We say that $m_1 \rightarrow m_2$ (or $m_1$ happened before $m_2$) if there is a directed path from message $m_1$ to $m_2$ in the graph.

Lemma A.1. The directed message graph produced by any execution of Algorithm 1 is a directed tree.

Proof. Acyclicity follows from the observation that each message sent by Algorithm 1 is unique, as identified by the unique two-tuple timestamp assigned in lines 11 or 23, thus the directed message graph is a directed acyclic graph (DAG). That DAG is also a tree because each message is caused by at most one other message: a CBGP message is caused by either a unique CBGP message (line 13) or the original GR (line 25) and the original GR being the root of the DAG is not caused by any other message.

In the proofs below, we will use $tree(m)$ denote the subtree rooted at $m$ in the directed message tree produced by an instance of Algorithm 1 (which is well-defined because of Lemma A.1 just above).

A.1 Proof of safety

We introduce the following lemmas in order to prove safety (Theorem 3.1).

Lemma A.2. If a router receives a FIZZLE for a CBGP message $m_1$ that it sent, then $tree(m_1)$ is finite.

Proof. Suppose the execution of Algorithm 1 started at time 0 and the router in the lemma’s premise received the FIZZLE for $m_1$ at some (finite) time $t_1$. Further assume that the time since a router sends a FIZZLE until the corresponding neighboring router receives it is lower bounded by a constant time $c > 0$ (as would be the case in any real implementation of Algorithm 1). To show that $tree(m_1)$ is finite, we show that every path rooted at $m_1$ is finite as follows.

Consider the set caused($m_1$) of the immediate children of $m_1$, which is the set of messages sent by the recipient of $m_1$ in either the for loop on either line 10 (if $m_1$ were a GR message) or line 22 (if $m_1$ were a CBGP message). By inspection of Algorithm 1, a FIZZLE is sent only at two places, line 20 and line 35. In the former case (line 20), the causal message $m_1$ did not spawn any further child messages at the recipient of $m_1$, so caused($m_1$) is empty implying that $tree(m_1)$ is of unit size.

In the latter case (line 35), the recipient of $m_1$ must have previously remembered the set of caused messages caused($m_1$) in its local map (the sent[$E$][is($m_1$)] map in lines 12 or 24) and subsequently sent a FIZZLE for $m_1$ back to its sender when the sent map got emptied at the recipient (line 34), i.e., the recipient of $m_1$ received a FIZZLE for every message in caused($m_1$). By assumption, the recipient of $m_1$ must have received a FIZZLE for every message in caused($m_1$) by time no later than $t - c$.

Let caused($m_1$) recursively denote the set of messages caused by caused($m_1$) for $k > 1$. By repeating the above argument, all messages in caused($m_1$) must have received a corresponding FIZZLE from their respective recipients by time $t - kc$. By assumption, the algorithm began execution at time 0, so the depth of $tree(m_1)$ is at most $t/c$.

Lemma A.3. A router receives a FIZZLE for a CBGP message $m_1$ it sent only if for every message $m_2$ (sent by any router) such that $m_1 \rightarrow m_2$, the sender of $m_2$ had previously received a matching FIZZLE from the recipient of $m_2$.

Proof. The proof is by induction on the depth of the subtree $tree(m_1)$ rooted at $m_1$, which by Lemma A.2 is finite.

Consider a topological sort of $tree(m_1)$ with the root $m_1$ at level 0 and all the messages in caused($m_1$) at level $k > 0$. Let the depth or maximum level be $d$ (that is well defined because of Lemma A.2). The lemma’s claim is trivially true for a level $d$ message as it did not cause any further messages. Suppose that the claim is true for messages at all levels in $[i, d]$ (both inclusive) for some $1 < i < d$. Consider a non-leaf message $m$ at level $i - 1$ sent by a router for some root event $E$. By line 35, the recipient router can issue a FIZZLE for $m$ only if its sent[$E$] is empty, i.e., only if it has received a matching FIZZLE for every message in caused($m$), which completes the inductive step for level $i$, proving the lemma.

Lemma A.4. When converged($E$) is set to true at the root router, sent[$E$] is empty at every router.

Proof. There are two cases to consider.
Case 1: $\text{converged}(E)$ is set upon receipt of a GRC message
In this case, the root router will terminate before sending any CBGP message. No router will send any further CBGP messages because a CBGP message can only be sent in response to the receipt of a CBGP or GRC message.

Case 2: $\text{converged}(E)$ is set upon receipt of a FIZZLE.
By inspection of code, this case can happen only if the router receiving the FIZZLE is the router that received the root cause GRC message. $\text{converged}(E)$ is set at the root router only when its $\text{sent}[E]$ map is empty (line 7). By inspection (line 31), an entry in $\text{sent}[E]$ is removed only if the matching FIZZLE is received. Thus, the root router must have received a matching FIZZLE for each message it sent including all the level 1 messages immediately caused by the root GRC message. By Lemma A.3, every sent message for $E$ (at any router) must have received a matching FIZZLE. By line 31, an entry in $\text{sent}[E]$ is removed if the matching FIZZLE is received. So, if every sent message for $E$ has received a matching FIZZLE, $\text{sent}[E]$ must be empty at all routers, proving the claim.

We complete the proof of safety as follows.

Theorem 3.1. SAFETY: If $\text{converged}(E)$ is true at the root cloud router that received the GRC, then no further CBGP message for $E$ will be received by any cloud router.

Proof. By Lemma A.4 above, $\text{sent}[E]$ is empty at every router at this point, so there is no in-propagation CBGP message for $E$ as every sent message at any router has already received the corresponding FIZZLE. The root router that received the GRC message terminates immediately after setting $\text{converged}(E)$, so it will not send any further CBGP messages for $E$. A CBGP message can be sent by a non-root router (line 26) only upon receiving an in-propagation CBGP message, so no non-root router will send any further CBGP messages either.

Technical note. If we assume safe BGP policies, the onerousness in Lemma A.2 would be unnecessary as the directed message tree would be bounded by definition. However, the proof of CIRCA’s safety (unlike the proof of its liveness immediately below) does not require the safety of the underlying BGP policies, so we chose the more onerous formal route to make that independence clear.

A.2 Proof of liveness
The liveness proofs below implicitly assume that routers in the CIRCA cloud either do not fail or employ a persistent write-ahead log to record all local state changes before sending messages based on those changes. (Unlike liveness, neither failures nor the availability of a persistent log impacts safety.)

Lemma A.5. If BGP policies are safe, every sent CBGP message eventually receives a matching FIZZLE if all cloud routers are available for a sufficiently long duration.

Proof. The proof of this claim is by induction and is similar in spirit to Lemma A.3. The assumed safety of BGP policies by definition bounds the size of the directed message tree. Consider a topological sort as before partitioning the tree into levels where level $k$ nodes are the messages in $\text{caused}^k(\text{GRC})$ where we have used GRC as a shorthand for the original GRC message that initiated Algorithm 1 at the root router. Let $d$ denote the depth of the tree.

The claim is true for level $d$ messages because of lines 19–26: a router receiving a CBGP message must either cause a new CBGP (the else block starting line 21) or issue a FIZZLE for the received CBGP (line 20). Level $d$ messages do not cause any further CBGP messages, so routers receiving them will issue a corresponding FIZZLE.

The inductive step is as follows. Suppose the claim holds for levels $i$ to $d$ for $1 < i < d$. Consider a level $i$ message $m$ with timestamp $ts$ received by a router that causes it to send the set $\text{caused}(m)$ of level $i$ messages. By the inductive assumption, each level $i$ message $m_1$ eventually receives a matching FIZZLE, and each such FIZZLE removes the corresponding entry from $\text{sent}[E][ts_1]$ at the router that received $m_1$ where $ts_1$ is the timestamp of $m_1$. Thus, when the recipient of $m$ has received matching FIZZLES for all level $i$ messages caused by the incoming level $i+1$ message $m$, its $\text{sent}[E][ts]$ map becomes empty triggering the matching FIZZLE for $m$, which completes the inductive step, proving the lemma.

Theorem 3.2. LIVENESS: Algorithm 1 eventually terminates if the underlying BGP policies are safe and all cloud routers are available for a sufficiently long duration.

Proof. To prove that Algorithm 1 terminates, i.e., $\text{converged}(E)$ is set at the root router, we consider two cases, the first of which as in the safety proof is trivial.

Case 1: No CBGP messages are sent.
In this case, the algorithm trivially terminates in line 7.

Case 1: CBGP messages are sent.
The proof of the theorem follows by observing that, by Lemma A.5, the root router must eventually receive a matching FIZZLE for all (level 1) CBGP messages caused by the root GRC (in line 26). At this point, its $\text{sent}[E]$ map must be empty for the following reason: Lemma A.3 implies that any level 2 or higher CBGP message sent by any router must have already fizzled, so any level 2 or higher CBGP sent by the root router must have also already fizzled and consequently already removed from its $\text{sent}[E]$ map. By line 32, an empty $\text{sent}[E]$ map causes Algorithm 1 to terminate.

Note that although the proofs above did not explicitly invoke the “all cloud routers are available for a sufficiently long duration” assumption, they implicitly relied on that in conjunction with a persistent write-ahead log in all claims with the word “eventually” in the proofs above.

---

3. A reminder that “safe” here means that a stable route configuration exists [13] and is unrelated to safety in Theorem 3.1.
A.3 Proof of efficiency

Algorithm 1 omitted a formal description of the third dissemination phase for a concise exposition of the key safety and liveness properties. For completeness, we codify the dissemination phase in event-action format as well. The root router that received the GRC message for event $E$, instead of exiting when $converged(E)$ becomes true (on line 33), initiates the dissemination phase by sending to itself the COMMIT message $(\text{COMMIT}, E, ts)$ where $ts$ is the timestamp of the original GRC message as assigned on line 9 in Algorithm 1. The event handler for a received commit message is below.

Dissemination phase of Algorithm 1

1: event recv($\langle$COMMIT, $E$, $ts \rangle$, $N$) $\triangleright$ upon receipt of a commit message from cloud router $N$ at cloud router $v(R)$
2: $p \leftarrow$ all affected prefixes in $\text{fizzled}[E][ts]$
3: send($\langle$FBGP, $E$, $p$, FIB($p$)), $R$) $\triangleright$ southbound FIB dispatch
4: for each $[N_1, p_1, ts_1] : \text{fizzled}[E][ts]$, do
5: \hspace{1em} send($\langle$COMMIT, $E$, $ts_1 \rangle$, $N_1$)
6: $\text{fizzled}[E][ts] \leftarrow \emptyset$
7: if $\text{fizzled}[E] = \emptyset$ then exit

THEOREM 3.3. EFFICIENCY: The root router (Any router) in Algorithm 1 detects convergence within at most $2\Delta(3\Delta)$ time and $2M(3M)$ messages where $\Delta$ and $M$ are respectively the actual convergence delay and number of messages incurred by the underlying distributed route computation protocol.

Proof. The convergence indicating COMMIT messages disseminated in Algorithm 1’s third phase by design follow the exact same paths as the CBGP messages in the directed message tree in the first (exploration) phase. This claim follows from the observation that when $converged(E)$ is set at the root router, Lemma A.4 implies that every entry ever previously added to the $\text{sent}[E]$ map (in lines 12 or 24) has already been removed, and by inspection of lines 31 and 30, every entry removed from the $\text{sent}[E]$ map is added to the $\text{fizzled}[E]$ map.

Assuming safety of the underlying BGP policies, every CBGP message eventually begets a matching FIZZLE in the reverse direction in the second (back-propagation) phase and a matching COMMIT in the same direction in the third (dissemination) phase, thus the number of messages in each phase is identical. The “actual convergence delay and number of messages incurred by the underlying distributed route computation protocol” are defined as those of the first phase alone.

The root router detects convergence at the end of the second phase when $converged(E)$ becomes true. Each router detects convergence when its $\text{fizzled}[E]$ map gets emptied, which necessarily happens for every router in the third phase by an inductive argument similar to that used in Lemma A.5.

Technical notes. The efficiency proof conservatively assumes that the second and third phases incur at most as much delay as the first phase. In a practical implementation, this assumption is more than true in that the first phase is likely to incur much more delay than the other two phases. The reason is that the first phase CBGP messages may carry a large number (hundreds or thousands) of prefixes resulting in large transfer sizes at each hop but the second and third phase messages are small control messages as they just need to convey the root cause event identifier and timestamp.

The dissemination phase as strictly codified above may result in any given router dispatching southward portions of the modified FIB entries for a single root cause event in multiple bursts (via line 3) in the third phase. It is straightforward to modify the protocol so as to either dispatch the entire set of modified FIB entries as a single message just before exit (on line 7) instead or send a termination marker southward just before exiting so that the ground incarnation can apply all modified FIB entries atomically. This modification may reduce transient inconsistencies in the ground forwarding plane, especially if cloud routers intermittently fail and pre-computed backup routing options are available to ground routers.

B Route computation equivalence

THEOREM 3.4. ROUTE COMPUTATION EQUIVALENCE: If for a sufficiently long period—(i) all ground routers can reach a CIRCA cloud datacenter and vice versa; and (ii) all cloud routers are available and can communicate in a timely manner—a pure CIRCA system ensures Route Computation Equivalence with any distributed route computation function that satisfies Eventually Consistent Convergence.

Proof. Let $D(S)$ denote any distributed route computation function. Suppose the initial network state is $S_0$ and a sequence of root cause events $[e_1, \ldots, e_n]$ (but no further events) occurs. Consider the following mutually exclusive and exhaustive set of event sequences: $\{(e_{11}, e_{12}, \ldots), [e_{21}, e_{22}, \ldots], \ldots, [e_{m1}, e_{m2}, \ldots]\}$ where $e_{ij}$ denotes an event at router $i$ with (local) sequence number $j$ and $m$ denotes the total number of routers, i.e., each sequence in this set is an ordered subset of events in the original event sequence all of which were detected by the same router in that local order.

The theorem follows from the following claims: (1) CIRCA eventually processes all root events; (2) CIRCA processes root events in an order consistent with the local order at ground routers; (3) processing the root events in any global order consistent with local orders results in the same final routing outcomes. The first two claims are straightforward: the first follows from the assumption of sufficiently long periods of cloud-ground reachability and cloud router availability; the second follows from line II.1 in the north-south protocol.

The last of the above claims is the non-intuitive one and needs a proof. The proof surprisingly follows from the assumption that the ground routing protocol ensures Eventually Consistent Convergence. Consider any reordering of the $f = [f_1, \ldots, f_n]$ of the original event sequence $E = [e_1, \ldots, e_n]$ that...
preserves local order. We claim that $S_0 | \mathcal{F} = S_0 | \mathcal{E}$, i.e., the network arrives at the same final state given any local-order-preserving reordering\(^3\) of an event sequence.

To see why, suppose events only consisted of link up and link down events for a given link. A link event will be detected and reported by one or both of the incident ground routers, say $A$ and $B$. No matter how many times the link goes up or down in a given sequence of events, in any local-order-preserving reordering of that sequence, both $A$ and $B$ will agree on the final state of the link. Note that the claim is not true in general for a non-local-order-preserving reordering, for example, if link $A - B$ went down and then came back up, but the last event was reported by $A$ and reported as a link down event, the final state of the network (and by consequence routing outcomes) will be different.

The complete proof follows from showing the third claim, namely that any sequentially consistent reordering of events produces the same final outcome, in a manner similar to above for other root cause events including node up/down events, link cost changes, as well as combinations of such events.

We conjecture that RCE given ECC as above holds for more general policy or other configuration changes at routers. It is straightforward to see that any set of configuration changes across distributed routers will preserve RCE if the final state is the same given any sequentially consistent reordering of those changes. However, it is unclear to us if this property always holds or to what extent it holds in practice. For example, if the value of a router configuration parameter is determined based on the value of another parameter at a different router, the final network configuration state may depend on the precise total order of all events, i.e., it is sensitive to different reorderings even if they are sequentially consistent.

### B.1 Practical considerations for RCE

The high-level CIRCA design may require or benefit from some adaptations in practice as summarized below in order to preserve route computation equivalence. A more detailed investigation of these is deferred to future work.

**IGP-awareness:** A root event such as a link cost change or, more generally, any intradomain root event that potentially affects interdomain routes at any router needs to be conveyed somehow to that router. In an intradomain routing protocol such as OSPF, link-state advertisements (LSA) accomplish that purpose. There are two natural design alternatives in CIRCA to accomplish the same: (1) piggyback the root cause event label in LSAs similar to CIRCA’s iBGP messages; (2) use iBGP to disseminate the root event network-wide within the domain. The former approach has the benefit of being largely decoupled from the intradomain routing protocol and works well for intradomain routing protocols that are global by design like link-state routing, but not so well for decentralized protocols like distance-vector routing wherein a router by design does not know the final outcome of the route computation until the decentralized computation completes. In practice, “full-mesh-iBGP-as-LSA” suffices to maintain route computation equivalence for any shortest path routing protocol or, more generally, any intradomain routing protocol that lends itself to a global implementation, i.e., one where an individual router can immediately compute its final intradomain routing outcome for any given change to global (intradomain) network state.

**Root cause event grammar:** Although the variable-length description \((\text{etype}, \text{eval})\) (refer §3.1.1 and Figure 4) is in principle sufficiently general to allow CIRCA to represent arbitrary changes to router configuration state, and parsing it is a purely intradomain concern, there may be value in minimally standardizing this representation across different router vendors for the sake of a reusable implementation. For example, the unix utility `diff` is a naive way to represent changes to router configuration files in a platform-agnostic manner. A more systematic vendor-specific or -neutral grammar for representing root cause events is conceivable.

**Domain consolidation:** Consolidating route computation for routers in the same domain by employing a single well-provisioned router server (similar in spirit to RCP [4]) instead of one-one mapping ground routers to virtual routers is likely to improve resource efficiency and thereby reduce the provisioning cost of route servers. The consolidated approach blurs the distinction between the abstract “single-router-AS” model and ASes in practice, arguably making it easier for operators to monitor and understand routing dynamics within their networks. Maintaining route computation equivalence requires that the consolidated server is guaranteed to computes a routing outcome that could have been computed by the corresponding ground protocol.