UCL DEPARTMENT OF COMPUTER SCIENCE

Research Note
RN/12/10

LOUP: The Principles and Practice of Intra-Domain Route Dissemination

05/10/12

Nikola Gvozdiev

Brad Karp

Mark Handley

Abstract

We propose the Link-Ordered Update Protocol (LOUP), a clean-slate dissemination protocol for external routes that does not create transient loops, makes stable route choices in the presence of failures, and achieves policycompliant routing without requiring any configuration. We demonstrate the practical scalability and correctness of LOUP through simulation and measurements of a Quagga-based implementation.

LOUP: The Principles and Practice of Intra-Domain Route Dissemination

Nikola Gvozdiev, Brad Karp, Mark Handley
University College London

Abstract

Network operators have gravitated toward building networks with a BGP-free core. They have done so for two main reasons. First, there was a point in history when it was thought to be difficult to do IP forwarding lookups quickly enough in large routing tables—a concern that has largely fallen by the wayside, thanks to algorithmic and hardware improvements. Second, there has been a perception in the networking community that the prevalent approach to scaling intra-domain dissemination of external routes, iBGP with route reflectors, is brittle: it requires careful configuration based on topology, policy, and heuristics. Under misconfiguration or topology changes, iBGP with route reflectors exhibits a variety of ills, including routing instability, transient loops, and routing failures. In this paper, we consider the intra-domain route dissemination problem from first principles, and show that these pathologies are not fundamental–rather, they are artifacts of iBGP. We propose the Link-Ordered Update Protocol (LOUP), a clean-slate dissemination protocol for external routes that does not create transient loops, makes stable route choices in the presence of failures, and achieves policy-compliant routing without requiring any configuration. We demonstrate the practical scalability and correctness of LOUP through simulation and measurements of a Quagga-based implementation.

1 Introduction

The past decade has seen many network operators adopt a Border Gateway Protocol-free (BGP-free) core for their networks, to an extent that educational materials on how to configure core routers in this fashion are readily available [5]. While complete statistics on the prevalence of a BGP-free core are hard to come by, an informal survey taken in 2010 suggested that at least 36% of responding network operators had replaced iBGP on their core routers with some form of label switching [14].

Two primary factors originally drove interest in a BGP-free network core. First, there was widespread concern that IP forwarding lookups would not keep pace with rapidly increasing link speeds and increasing forwarding table sizes. And second, there was a sentiment that internal BGP (iBGP), typically deployed with Route Reflectors (RRs), was an unsatisfactory way to disseminate externally learned routes within an autonomous system (AS). Much has been written about iBGP's susceptibility to persistent routing instability [11, 12], as well as its propensity for introducing transient forwarding loops and transient routing failures while disseminating a route update or withdrawal [11, 20]. Real-time traffic of the sort prevalent on today's Internet does not tolerate such transient loops or failures well; Kushman *et al.* [18] note that periods of poor quality in VoIP calls correlate closely with BGP routing changes. Even a BGP-free core does not entirely eliminate iBGP's role in intra-AS route dissemination, nor the associated pathologies: border routers (BRs) must still exchange routes with one another with iBGP.

Is the move to a BGP-free core warranted? We argue that it is not. Thanks to algorithmic and hardware advances, the speed of forwarding lookups is no longer a worry. More interestingly, though, the routing instability, transient loops, and transient black holes that often occur when disseminating a route update (or withdrawal) learned via eBGP within an AS are not fundamental. Rather, they are side-effects of the way in which the iBGP protocol, and in particular, the iBGP protocol with RRs, happens to disseminate routes.

We present the Link-Ordered Update Protocol (LOUP), a simple and robust intra-AS route dissemination protocol that introduces neither transient loops nor black holes while propagating an update. LOUP avoids these pathologies by reliably disseminating updates hop-by-hop along the *reverse forwarding tree* from a BR. We are not the first to observe that careful attention to the details of route propagation can eliminate transient anomalies. DUAL [8], the loop-free distance-vector interior gateway protocol (IGP), explicitly validates before switching to a next hop that doing so will not cause a forwarding loop. And Consensus Routing [17] augments eBGP with Paxos agreement to ensure that all ASes have applied an update for a prefix before any AS deems a route based on that update to be stable. LOUP uses comparatively light-weight mechanisms (*i.e.,* forwarding in order along a tree) to avoid transients during route dissemination within an AS.

Our contributions in this paper include:

- a first-principles exploration of the dynamics of route dissemination, including how known protocols do dissemination and the trade-offs they make
- invariants that, when maintained during route dissemination, avoid transient loops when a single update to

a prefix is introduced
- LOUP, a route dissemination protocol that enforces these invariants using ordered dissemination of log entries along a tree
- an evaluation in simulation that demonstrates the correctness and scalability of LOUP on a realistic Internet Service Provider topology
- measurements of an implementation of LOUP for Quagga that show LOUP scales well in memory and computation, so that it can handle a full Internet routing table and high update processing rate

2 Intra-AS route dissemination

Internet routing is composed of three components: *External BGP* (eBGP) distributes routes between routing domains and is the instrument for policy routing. An *Interior Gateway Protocol* (IGP) such as OSPF or IS-IS keeps track of reachability within a routing domain. Finally, *Internal BGP* (iBGP) distributes external routes received by border routers (BRs) both to all other BRs, so they can be redistributed to other domains, and also to all participating internal routers. In this paper we are primarily concerned with iBGP: when a route changes somewhere out there in the Internet, how does this change propagate across a routing domain?

When a BR receives a route change from a neighboring routing domain or *Autonomous System* (AS), it sanity checks it and applies a policy filter. This policy filter can drop the route, or it can modify it. Common modifications include setting the Local Preference attribute (akin to a priority field) and setting Community attributes which can be used to determine how the route is redistributed to other ASs by other BRs.

After policy filtering, the BR runs its decision process, determining whether it prefers this route to other routes it may hold for the same IP address prefix. The decision process is specified in the BGP standard, and consists of successive rounds of eliminating candidate routes based on different criteria until only one remains. First in the decision process is *Local Preference*, so configured policy trumps all else. Lower down the list come AS Path length, and below that IGP distance (the distance to the BGP Nexthop - usually either the announcing BR itself, or its immediate neighbor in the neighboring AS).

When a BR receives a route announcement for a new prefix, if it is not dropped by policy, the BR distributes it to all the other routers in the domain, so they can reach this destination. If a BR already has a route to that prefix, the new route is only sent to the other routers if the BR prefers the new route. Similarly if a BR hears a preferred route from another BR to one it previously announced, it will withdraw the previously announced route.

Having decided to announce or withdraw a route, it is important to communicate the change reliably and quickly to the rest of the AS. BGP routing tables are large - currently over 400,000 prefixes - and multiple BRs can receive different versions of each route. Periodic announcement of routes doesn't scale well, so dissemination needs to be reliable - once a router has been told a route, it will hold it until it is withdrawn or superseded. Route dissemination also needs to be fast - otherwise inter-AS routing can take a long time to converge. Ideally it would also be consistent - a route change from far across the Internet can be received by multiple BRs in a short time period, and their uncoordinated propagation of route changes across the domain can itself cause disruption.

The simplest way to disseminate routes across an AS is full-mesh iBGP, where each router opens a connection to every other router in the domain (Fig. 1a). When an update needs to be distributed, a BR just sends it down all its connections. TCP then provides reliable in-order delivery of all updates to each router, though it provides no ordering guarantees between different recipients.

In practice, few networks run full-mesh iBGP. The $O(n^2)$ TCP connections it requires dictate that all routers in a network must be reconfigured whenever a router is added or retired, and every router must fan out each update to all $n-1$ peers causing a load spike with associated processing delays. Most ISPs use iBGP route reflectors (RRs). These introduce hierarchy; they force propagation to happen over a tree[1] (Fig. 1b). Updates are sent by a BR to its reflector, which forwards them to its other clients and to other reflectors. Each other reflector forwards on to its own clients.

Route reflectors significantly improve iBGP's scaling, but they bring a range of problems all their own. In particular, each BR now only sees the routes it receives directly via eBGP and those it receives from its route reflector. Thus no router has a complete overview of all the choices available, and this can lead to a range of pathologies, including persistent route oscillations[9].

ISPs attempt to avoid such problems by manually placing route reflectors according to guidelines that say "follow the physical topology"; not doing so can cause suboptimal routing[20]. Despite these issues, almost all ISPs use route reflectors and, with conservative network designs, most succeed in avoiding the potential pitfalls.

The Rise of the BGP-free Core

In recent years many ISPs have deployed MPLS within their networks, primarily to support the provisioning of VPN services. MPLS also allows some networks to operate a BGP-free core.

[1] actually, often two overlaid trees for redundancy

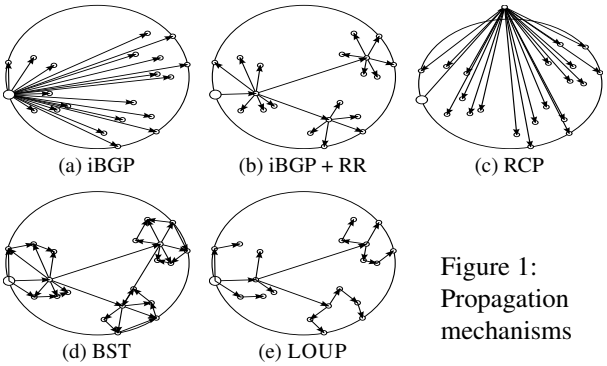

Figure 1: Propagation mechanisms

An MPLS network with a BGP-free core functions in the same way as BGP with route reflectors, except that only BRs are clients of the RRs. An internal router that only provides transit services between BRs does not need to run BGP; instead the entry BR uses an MPLS label-switched path to tunnel traffic to the exit BR. A protocol such as LDP [2] is then used to set up a mesh of MPLS paths through the AS from each entry BR to each exit BR. In such a network, internal routers do not have external routes in their routing tables, so they cannot reach any external networks, except by first following a default route to a BR that does know BGP routes.

A BGP-free core needs careful configuration, and depends on additional protocols such as LDP or RSVP-TE for basic connectivity, adding complexity. Why would an ISP choose to remove iBGP from internal routers in this way? It isn't for traffic engineering reasons, as MPLS traffic engineering can be used to incrementally optimize a natively-routed AS.

One potential advantage of a BGP-free core is that core routers need only maintain IGP routes, rather than the 400,000 or so prefixes in a full routing table, though they must also hold MPLS state for a subset of the $O(n^2)$ label-switched paths. In general though, due to improvements in hardware and forwarding algorithms [23], the overall size of routing tables is not the problem it was once thought to be [6]. Backbone routers tend to be *more* capable than border routers; a core router than cannot forward at line rate with a full routing table is not likely to sell well. If there is any scaling issue, it is unlikely to be with routing table size, but with churn of the FIB forwarding table. Even this is unlikely to be an issue, so long as it is constrained to updating existing forwarding entries; additions and deletions may be more costly if they require updating tree-based FIB structures [6].

Another potential advantage of a BGP-free core is that it reduces the number of clients of iBGP route reflectors. This has the potential to reduce both processing load on the RRs and the potential for undesirable configuration errors resulting in instability. In short, the fewer iBGP routers you have, the less likely you will be bitten by the known limitations of iBGP and route reflectors.

Finally, although it doesn't seem to be well understood by ISPs, the use of route reflectors can result in transient forwarding loops every time an external route changes. This is because the TCP connections used by route reflectors do not impose any inter-client ordering on who learns each update when. While such loops are unlikely to persist for long, any forwarding loop can cause non-trivial packet loss and jitter, not just for the looping traffic but also for traffic sharing the same links. Although MPLS may reduce the prevalence of such loops, it cannot prevent them - those transient loops that do occur will traverse the whole network between two (or more) BRs.

Thus the drive towards networks with a BGP-free core seems to be driven by obsolete concerns about routing table size and by undesirable properties of iBGP with route reflectors. Our goal is to revisit the role played by iBGP, and demonstrate that iBGP's limitations are not fundamental. We will show that alternatives to iBGP are possible with the following properties:

- Not susceptible to configuration errors.
- Stable under all configurations.
- Not prone to routing hot spots.
- Low FIB churn, both in BRs and in internal routers.
- Propagate no more than the minimum required number of changes to eBGP peers.
- Free of transient loops and unnecessary black holes.
- Play nicely with MPLS traffic engineering, but not mandate its use.

3 Dissemination Mechanisms

We now examine alternative route dissemination mechanisms that cast light on what is achievable.

3.1 Persistent Route Oscillations

With eBGP, inconsistent policies between ISPs can lead to persistent BGP oscillations. These can be avoided if BGP policies obey normal autonomous systems relations ("obey AR") [7]. Essentially this involves preferring customer routes to peer or provider routes, and that the graph of customer/provider relationships is acyclic. However, even when AR is obeyed, BGP's MED attribute can result in persistent iBGP route oscillations [10].

Briefly, MED allows an operator some measure of control over which link traffic from him provider takes to enter his network. Unfortunately the use of MED means that there is no unique lexical ordering to alternative routes. The decision process essentially takes two rounds; in the first routes received from the same ISP are compared, and the ones with higher MED are eliminated; in the second, the remaining routes are compared, and an overall winner is chosen. Thus route A can eliminate route B in the first round, then lose in the second route

to route C. However, in the absence of route A, route B may win the second round. Compared pairwise, we have $A > B$, $B > C$, and $C > A$. To make the correct decision, routers must see all the routes, not a subset of them.

Route reflectors hide information; only the best route is passed on. This interacts poorly with MED, resulting in persistent oscillations [19]. Griffin and Wilfong prove that so long as policies obey AR, full-mesh iBGP will always converge to a stable solution [10]. The same is not true of route reflectors or confederations. To avoid iBGP route oscillations, it is sufficient to converge to the same routing solution that would be achieved by full-mesh iBGP, and we adopt this as a goal.

3.2 Transient Loop Avoidance

Whenever iBGP disseminates routes, its TCP connections ensure the in-order arrival of updates at each recipient. However as each recipient applies the update as soon as it can, this results in an arbitrary ordering *between* recipients, causing transient loops and black holes until all the routers have received and processed the update. Route reflectors impose a limited ordering constraint (RR clients receive a route after their RR processes it), but except in trivial non-redundant topologies this is insufficient to prevent loops and black holes.

One way to avoid both loops and persistent oscillations is to centralize routing (Fig. 1c). When an external update arrives the BR first sends it to the routing control platform (RCP [3]), which is in charge of running the decision process for the whole domain and distributing the results to all routers. As the RCP router has full knowledge, loops can be avoided by applying updates in a carefully controlled order. However, to do so requires a synchronous update approach which, given the RTTs to each router, will be slower than distributed approaches.

In the case of IGP routing, it is well known how to build loop-free routing protocols. DUAL [8] is the basis of Cisco's EIGRP, widely deployed in enterprise networks, and uses a provably loop-free distance vector approach. DUAL is based on several observations:

- If a metric change is received that reduces the distance to the destination, it is always safe to switch to the new shortest path. This is a property of distance vector routing; if the neighbor sending the change is the new nexthop, it must have already applied the update, and so must be closer to the destination. No loop can occur.
- If an increased distance is received, the router can safely switch to any neighbor that is closer than it previously was from the destination. These are known as feasible routes. They are safe because no matter how far away the router becomes after the update, the router still never forwards away from the destination.
- In all other circumstances, a router receiving an increased distance cannot safely make its own local decision. DUAL uses a *diffusing computation* to make its choice. It locks the current choice of nexthop and queries its neighbors to see if they have a feasible route. If they do not, they query their neighbors, and so on until the computation reaches a router close enough to the destination that it can make a safe choice. The responses spread out back across the network, activating routes in a wave that spreads out from the destination, and so avoiding loops.

The iBGP route dissemination problem is different from that solved by DUAL, as no incrementally-updating distance metrics are involved. Instead, for each prefix routers must decide between alternative external routes as those routes propagate across the network. The routes themselves do not change; rather to avoid loops we must either control the order in which route changes are received or the order in which they are applied.

3.2.1 Wavefront Propagation

DUAL performs hop-by-hop flooding of route changes, accumulating metric changes along the way. BGP route dissemination can also be performed using hop-by-hop flooding (Fig. 1d), with each router sending the messages it receives to all neighbors. Flooding needs to be done over one-hop reliable sessions to ensure messages are not lost. This is the approach taken by BST [16]. Flooding imposes a topological ordering constraint, guaranteeing that at all times, a contiguous region of routers have processed an update. Essentially an update propagates out across the domain as a *wave-front*; this is a necessary (though not sufficient) condition to avoid transient loops. None of the other mechanisms above have this property.

To see why this condition is not sufficient, even in the presence of a new route being propagated (the equivalent of DUAL receiving an improved route), consider Fig. 2. BR B had previously received a route to prefix P, and distributed it to all the routers in the domain. BR A then receives a better route to P, and this is in the process of flooding across the domain, forming a wave-front ① which is flowing outwards from A. All the routes in the light-gray region now forward via A; the remainder still forward via B. Unfortunately flooding does not ensure that the wave-front remains convex - that a forwarding path only crosses the wave-front once. As a result transient loops ③ can occur.

Fig. 5 shows one way that such non-convexity can occur. Initially all routers forward to some prefix via B_2, but then B_1 receives a better route. Link 1-2 would not be normally used because of its high metric. If, however, router 1 floods the update from B_1 over this link, then receiving router 2 may direct traffic towards router

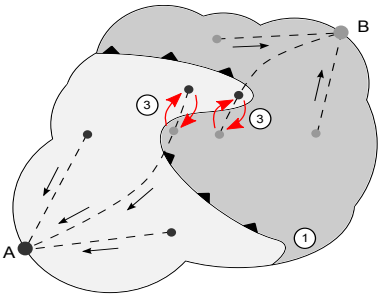

Figure 2: Transient loop when flooding an update.

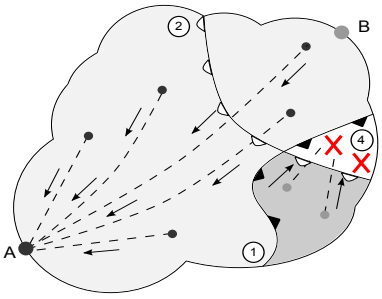

Figure 3: Racing update and withdrawal cause black hole

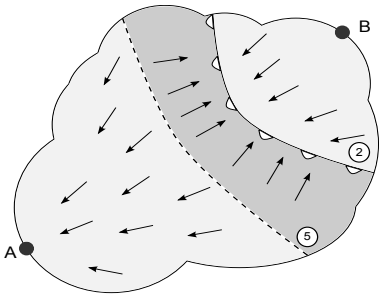

Figure 4: Withdrawal causes loop when alternate exists

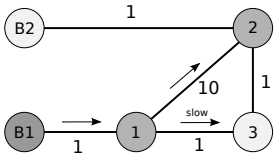

Figure 5: A simple topology exhibits looping

3 which is on the forwarding path to B_1. As router 3 has not yet heard the update, it will direct traffic towards B_2 via router 2, forming a loop. This loop will clear eventually when router 3 hears the update. The update may get delayed due to network conditions (congestion, packet loss) or, more likely, due to variable CPU processing delays.

3.2.2 Reverse Forwarding Tree Dissemination

BST's loops occur when a new route is propagating from one BR, and that route is preferred to a pre-existing route from another BR. A sufficient condition for avoiding such loops is that *a router does not adopt the new route until the next hop for that route has also adopted the route*. With routes that improve, this is essentially what DUAL's distance vector mechanism ensures. This condition transitively guarantees that a packet forwarded using the new state will not encounter a router still using the old state. One way to meet this condition for BGP route dissemination is for a router to only forward a route to routers that will forward via itself. Thus routes flow from a BR along the reverse of the forwarding tree that packets take to get to that BR.

In this paper we discuss a new routing mechanism we call Link-Ordered Update Protocol (LOUP) that is built around this principle . It works by propagating messages over a hop-by-hop tree (Fig. 1e). Unlike the Route Reflector tree, LOUP uses one tree per BR, rooted at that BR. This tree is dynamically built, hop-by-hop, and follows the underlying IGP routes to get to that BR from everywhere in the domain. The hop-by-hop nature preserves the wave-front property, and by distributing down the reverse of the forwarding tree (RFT), it adds additional desirable ordering constraints that eliminate transient loops of the form detailed above when improved routes are being disseminated.

3.2.3 Sending Bad News

Most of the mechanism in DUAL concerns how to process bad news - a link went down or a metric increased. In many ways DUAL solves a more difficult problem than that of BGP dissemination. Consider the case where a destination becomes unreachable. DUAL must solve the distance-vector count-to-infinity problem to remove routing state, whereas LOUP only needs to remove the old BGP route which it can do by sending a withdrawal message along the RFT tree from the BR that originated the route. This implicit ownership of a route by its originating BR simplifies the problem.

However, sending bad news is never as simple as sending good news. If a router receives a withdrawal (even over the RFT), it cannot just pass it on and locally delete the route from its FIB. If it does, a transient loop may result. Consider what happens when all routers hold more than one route to the same destination prefix. Typically this is when routes tie-break on IGP distance: more than one BR originates a route, but they are all equally preferred. Each router chooses the route to the closest exit, with some routers making one choice and some another. Such hot-potato routing is common when two ISPs peer in multiple places.

A loop occurs when one of those routes is withdrawn (Fig. 4). The routers behind withdrawal wave-front ② have already switched to an alternative route via A. Routers further away have not yet heard the withdrawal and still forward to B. Traffic loops at wave-front ②.

3.2.4 Activating withdrawals: "Tell me when..."

Whereas the condition for activating a new route ("good news") is simple (the next hop also knows the route), the general condition for ceasing to using a withdrawn route is slightly more complex - no router must be using this route to forward to you. When receiving a withdrawal, a router can pass the message on to its children on the RFT,

routes before event	event that occurs							
	withdraw X	withdraw Y	announce $Z = X$	announce $Z < X$	announce $Z > X$	X gets worse than Y	Y gets better than X	X gets better
no route	n/a	n/a	fwd Z on tree			n/a	n/a	n/a
X	any order	n/a	fwd Z on tree	no-op	fwd Z on tree, withdraw X ⋆	fwd X on tree	n/a	fwd X on tree
$X = Y$	only apply withdrawal when parent does †		fwd Z on tree	no-op	fwd Z on tree, withdraw X,Y ⋆	withdraw X †	fwd Y on tree, withdraw X ⋆	fwd X on tree, withdraw Y ⋆
$X > Y$	fwd Y on tree ‡	no-op	fwd Z on tree	no-op	fwd Z on tree, withdraw X ⋆	withdraw X fwd Y on tree ‡	fwd Y on tree, withdraw X ⋆	fwd X on tree

Table 1: Effect of route changes on network with zero, one or two existing routes X and Y.

but not yet apply it. It then asks its RFT children to tell it when they stopped using the losing route. We call this request a *tell-me-when* message; it is a separate single-hop message, sent over the already-open TCP session to the neighbor. Upon receiving a reply a router knows that it is now safe to activate the withdrawal because none of its children are using the route.

In Fig. 4, the withdrawal is not applied until it reaches ⑤. The first hop to the left of ⑤ triggers the reverse activation of the withdrawal by sending a reply. This causes the next router to apply the withdrawal, which causes it to reply, and so on back down the RFT tree from *A*.

3.2.5 Ordering and Triggered Races

It is easy to show that for single updates, so long as the IGP is loop-free, the combination of propagating new updates along the RFT and reverse activation of withdrawals using the responses to *tell-me-when* messages is sufficient to avoid transient loops. Unfortunately iBGP is not quite as simple as this. In particular, a withdrawal can trigger an announcement of an alternative route, or an update can trigger the withdrawal of the previously best route. Such triggered updates can race each other, so we also need to consider *inter-origin ordering*.

There are two cases to consider. First, a route can be withdrawn, and another route must take over. Second, a new winning route can trigger the withdrawal of the previous winning route. In both cases the order in which the two route changes are applied can lead to transient loops or black holes. As two (or more) changes are involved, distribution down the RFT, which only ensures intra-origin ordering is not sufficient.

For LOUP to be free from unwanted transient effects, it must enforce both inter and intra-origin ordering.

Table 1 shows all the possible external route changes hat can occur, and their effect on a network containing either zero, one, or two existing routes for a prefix. $X > Y$ indicates route X beats route Y in the decision process at all routers; $X = Y$ implies the two routes tie-break on IGP distance - some routers use one exit and some use the other. We assume changes are propagated down the RFT tree ensuring intra-origin ordering is maintained. The entries highlighted in gray show all the cases where intra-origin ordering is insufficient, and races resulting in transient loops or unnecessary black holes can occur.

Although there are multiple causes, only three distinct ordering problems emerge:

⋆ **UW-race**: an update/withdraw race

† **W-order**: a withdrawal ordering problem

‡ **WA-race**: an announce/withdraw race

The *W-order* case is the withdrawal ordering problem discussed above, and solved using *tell-me-when* queries.

For a *UW-race* to occur, the network must already know one route to the destination when another better route arrives via a different BR. This is shown in Fig. 3. A new better update is flowing out from BR *A* and is depicted by wave-front ①. However, this time it has already reached BR *B*. *B* now withdraws its own route because it has been obsoleted by the new route. The withdraw spreads out as wave-front ②, and the scene is set for a race. Some routers have not yet heard the new update. Because there is no enforced inter-origin ordering, those routers are exposed. Routers ④ hear the withdrawal before the update, apply it and temporarily blackhole any traffic destined for that prefix.

For a loop of type *WA-race* (‡) to happen, two or more BRs must hold different routes to the same prefix, with one route being preferred. In this case, the BRs holding the less preferred routes will have withdrawn them, and all the other routers will only hold the best route.

When the best route is withdrawn a *WA-race* loop can occur. The withdrawal reaches the BR holding the route that previously lost, and this is now re-announced. The withdrawal and announcement now race. This is almost the inverse of Fig. 3: some routers hear the withdrawal first and some hear the announcement first. Routers hearing the announcement first will not apply it, as they still prefer the better route, but they can forward the announcement to routers that have already heard the withdrawal. Packets can then loop between the two.

Interestingly, we observe that *UW-race* and *WA-race* cannot occur with BST. Hop-by-hop reliable flooding ensures that no router ever receives a triggered update before it receives the triggering event itself. Succinctly, it ensures that *cause* always happens before *effect*. By

Forwarding updates along the RFT tree can ensure intra-origin ordering, at the expense of inter-origin ordering.

3.3 General Solution for Loop Avoidance

The simplest way to get both inter-origin and intra-origin ordering is to separate route change distribution from route change activation. To distribute changes, we can use BST-like flooding. This ensures no router ever hears a triggered change before it hears the triggering change. Although such route changes are propagated on to neighbors, they are not activated until their activation condition is met.

In the case of activating a new (or improved) route, the basic activation condition is that the route has been also received from its parent over the RFT.

In the case of activating a withdrawal, we can send *tell-me-when* queries to all routers that were downstream on the RFT with respect to the route being withdrawn. This guarantees no loop can occur between this router and its downstream neighbors.

Such a guarantee ensures a *WA-Race* cannot cause a transient black hole - even though the upstream router may have only heard the withdrawal and the downstream router may have heard both the withdrawal and the triggered announcement, no loop can occur because the upstream router cannot activate its withdrawal before the downstream router does.

We note that an update where a previously-winning route becomes less preferred is also "bad news". We need to use *tell-me-when* requests when switching away from such a route to avoid potential Update-Announce races (a worsening route triggers announcement of an alternative) that are equivalent to a *WA-Race*.

Finally, a router should not activate a withdrawal if its only alternative route is an update that has not yet been activated. This avoids unnecessary black holes in the *UW-Race* condition.

These rules are sufficient to prevent transient loops while distributing both updates and withdrawals as they provide both intra-ordering and inter-ordering.

3.4 Optimized Solution: LOUP

Although the general solution above is sufficient to avoid transient loops and easy to understand, it is a little heavy-weight. It floods many unnecessary messages and can, in some cases, needlessly delay switching to a working route when an old route is withdrawn.

If we only send updates and withdrawals over the RFT, we can eliminate unnecessary messages, but we will need to add specific additional mechanisms to deal with inter-origin races. Let us examine each condition in turn.

3.4.1 *W-order*† and Targeted tell-me-when

The basic *tell-me-when* message described above was sent to all downstream neighbors and a response was required before applying the withdrawal. We can be more targeted in its use because loops are only a problem if a router actually has an alternative route. If it does, it can run the decision process to decide the new exit router (*A* in Fig. 4), determine the neighbor towards that exit, and only send the *tell-me-when* request to that neighbor. Upon receiving a reply a router knows that it is now safe to apply the withdrawal because its downstream neighbor (and transitively, all routers between it and the new exit point) is already using either the alternative.

3.4.2 *UW-race*⋆ and Predicated Withdrawals

Flooding updates can avoid the transient black hole in Fig. 3, but we would prefer a cheaper solution. In LOUP, the BR that originates such a withdrawal attaches a predicate to it to prevent anyone applying the withdrawal without having first seen the update that caused it. A predicated withdrawal is stored by routers as it travels down the tree, but is not applied until the predicate is met. In this case, the predicate specifies that the update that caused the withdrawal must have been heard. This ensures that no router can end up without a route. For example in Fig. 3 all the routers in ④ will wait until ① reaches them and only then withdraw the route. Note that forwarding towards *B* while waiting for the predicate to be satisfied cannot cause a loop because the traffic will always reach routers that have heard the update from A, so will still reach the correct exit router.

3.4.3 *WA-race*‡ and Decision Modifiers

Targeted *tell-me-when* only works if a router already has an alternative route. If one route is globally better, other alternatives will have been withdrawn from the domain, leading to the *WA-race* when the best route is withdrawn causing the next-best route to be re-announced. One option is to send *tell-me-when* to all downstream routers in this case, but a more elegant solution is possible.

In LOUP we ensure that the re-announcement will always win, regardless of whether or not a recipient has heard the withdrawal for the previously-best route. When the new-best route is re-announced it is tagged with a decision modifier. This causes all recipients to exclude the withdrawn route from the set of routes they use when running the decision process. They behave as if they have already heard the withdrawal, mimicking the effect of flooding. Since everyone will reliably receive the withdrawal via its RFT at some point, this approach is safe.

3.4.4 LOUP Completeness

Table 1 can be extended to cover three pre-existing routes (and by inference, an arbitrary number) by adding four more rows to cover possible partial orderings of three route preferences. When we do this, we find that three of the four reduce to cases already covered in the table. The remaining row ($X < Y < Z$) results in a more complex chain when Z is withdrawn: both X and Y may be re-announced, and then X is withdrawn again. The two announcements racing cause no problem, but there is the potential for both *WA-race* and *UW-race* simultaneously. Existing LOUP mechanisms are sufficient to handle this.

The table only covers races that are *inherent* in route dissemination, as it only covers a single triggering event. Both races can also occur due to unlucky timing of external events. Decision modifiers can prevent loops in an externally triggered *WA-race*, but no router has enough information to specify the predicate for an externally triggered *UW-race*, where the old route is externally withdrawn before the new route has fully propagated. There is no loop, but a transient black hole may result.

In using solutions targeted at specific known races rather than the more powerful and heavyweight general solution, we risk having missed potential unknown races. In general, the targeted *tell-me-when* mechanism makes withdrawals much safer, but not as safe as the full *tell-me-when* mechanism which mirrors DUAL's diffusing computation. At the moment we cannot guarantee there is *no* external sequence of very closely timed events that can cause a transient loop. We have searched for such sequences in our simulator, running more than 100,000 randomly generated scenarios. The only transient loop we found was when a BR sends an update immediately followed by a withdraw within a link-RTT, and these both cross in transit with a another update. The full *tell-me-when* mechanism would prevent this case, but it would also be prevented by BGP's existing MRAI timer which prohibits such fast churn. We can however show that LOUP always reaches the same solution as full-mesh iBGP, and so is free from persistent oscillations if the domains obey AR.

3.5 Freedom from Configuration Errors

Full-mesh iBGP requires all the peerings are configured. The configuration is simple, but all routers must be reconfigured whenever routers are added or removed. Route reflectors and confederations add configured structure to an AS, and require expert knowledge to follow heuristics to avoid sub-optimal routing or persistent oscillations. Running a BGP-free core adds improves iBGP's scaling somewhat, at the expense of requiring additional non-trivial mechanisms just to route traffic across the network core. Why should it be so complex to simply to route traffic to external destinations that are already known by an ASes BRs? All this configuration significantly increases the likelihood of outages caused by configuration errors.

Hop-by-hop dissemination mechanisms such as BST and LOUP can be configuration-free. All that is required is to enable the protocol. ISPs don't always want auto-configuration though, if the price of it is traffic concentration or routing hotspots without any controls to relieve them. We will need to show that LOUP does not cause such hotspots. In any event, the default view provided can always be tuned on an as-needed basis using MPLS-TE, OSPF metrics, or a range of other possibilities.

4 Protocol Design and Implementation

We now describe the instantiation of ordered update dissemination along an RFT in the LOUP protocol. The protocol includes two main aspects: how to build the RFT, and how to disseminate updates along the RFT reliably despite topology changes.

4.1 RFT Construction

Each LOUP router derives a unique ID (similar to BGP-ID) that it uses to identify routes it originates into the AS. LOUP routers periodically send single-hop link-local-multicast Hello messages to allow auto-discovery of peers. A Hello contains the sender's ID and AS number. Upon exchanging Hellos containing the same AS number, a pair of LOUP routers establish a TCP connection for a peering. All LOUP protocol messages apart from Hellos traverse these TCP connections, and are sent with an IP TTL of 1.

A LOUP router must know the IDs of all LOUP routers in its AS to build and maintain the RFT. This list is built by a gossip-like protocol that operates over LOUP's TCP-based peerings. Essentially, a LOUP router announces the full set of LOUP router IDs it knows to its neighbors each time that set grows (and to bootstrap, it announces its own ID when it first peers with a neighbor). These gossip messages need not be sent periodically, as they are disseminated reliably with TCP. LOUP routers time out IDs from this list upon seeing them become unreachable via the IGP.

The RFT rooted at a router X is the concatenation of the forwarding paths from all routers to X—the *inverse* of the relevant adjacencies in routers' routing tables. To build and maintain the RFT, each LOUP router periodically sends each of its neighbors a Child message. LOUP router Y will send its neighbor X a Child message stating, "you are my parent in the RFT for this set of IDs." This set of IDs is simply the set of all IGP-learned destination

IDs in Y's routing table with a next hop of X. Upon receiving a Child message on interface i, LOUP router X subsequently knows that it should forward any message that *originated* at any ID mentioned in that Child message down the appropriate RFT on interface i.

4.2 Reliable RFT Dissemination

An *origin* LOUP router that wishes to send an update (*e.g.,* a BR injecting an update received over eBGP) sends that update to all routers in its AS over the RFT rooted at itself. LOUP routers forward such updates over their one-hop TCP peerings with their immediate neighbors on the appropriate RFT. During a period when no topology changes occur in an RFT, TCP's reliable in-order delivery guarantees that all updates disseminated down the RFT will reach all routers in the AS.

When the topology (and thus the RFT) changes, however, message losses may occur: if the distance between two routers that were previously immediate neighbors changes and exceeds a single hop, the IP TTL of 1 on the TCP packets LOUP sends over its peerings will cause them to be dropped before they are delivered. For RFT-based update dissemination to be reliable under topology changes, then, some further mechanism is needed.

To make update dissemination over the RFT robust against topology changes, the LOUP protocol structures updates as a *log.* Each router maintains a log for each origin. A log consists of a series of operations, each with a sequence number, ordered by these sequence numbers; this sequence number space is per-origin. An operation may either be a route Update or a route Withdrawal. When a LOUP router receives an operation for dissemination over the RFT on a TCP peering with a neighbor, it only accepts the operation and appends it to the appropriate origin's log if that operation's sequence number is one greater than that of the greatest sequence number of any operation already in that origin's log. That is, a router only accepts operations from an origin for RFT dissemination in contiguous increasing sequence number order.

Should a LOUP router ever receive an operation for RFT dissemination with a sequence number other than the next contiguous sequence number, or should a temporary partition occur between erstwhile single-hop-neighbor routers, LOUP may need to recover missing operations for the origin in question. A LOUP router does so by communicating the next sequence number it expects for each origin's log to its current RFT parent. LOUP includes this information in Child messages, which routers send their parents for RFT construction and maintenance, as described above. Should an RFT parent find that it holds operations in a log that have not yet been seen by its RFT child, it forwards the operations in question to that child.

Provided that the topology within an AS remains stable long enough for LOUP to establish parent-child adjacencies with its periodic Child messages, LOUP's single-hop, IP TTL 1 TCP connections coupled with its log mechanism guarantee reliable dissemination of operations down the RFT, even when topology changes temporarily disrupt the RFT.

When a BR wishes to distribute a route Update or Withdrawal, it acts as an origin: it adds this operation to its log with the next unused sequence number, and sends it down the RFT. As nodes receive the operation, they apply it to their RIBs. The end effect is the same as that of full-mesh iBGP because the origin BR disseminates its Update or Withdrawal to every router in the AS, just as full-mesh iBGP does.

We note that a log for an origin will only contain a single operation per prefix. When a Withdrawal of a prefix follows an Update for that prefix, the recipient can immediately delete the earlier Update and store the later Withdrawal.[2] The recipient must store the Withdrawal because a child on the RFT may subsequently need to be sent the Withdrawal (if its Child message indicates that it has not yet seen it).

5 Evaluation

We evaluated LOUP to examine both its correctness and scalability. To be correct LOUP must:

- Always converge to the same solution as full-mesh iBGP. This guarantees no persistent oscillations if eBGP policy obeys AR.
- Not create transient loops, so long as the underlying IGP topology is loop-free.

We assess scalability by looking at:

- How is the processing load distributed between routers?
- How are FIB changes distributed? Do LOUP routers update the FIB more than iBGP routers?
- How much churn does LOUP propagate into neighboring ASes?
- What is the actual cost of processing updates? Can LOUP handle bursts of updates quickly enough?
- Can the implementation hold the global routing table in a reasonable memory footprint? LOUP does not hide information, so how well does it compare to BGP with RRs?

[2]The absence of the deleted earlier Update in the log, noticeable from the gap in sequence numbers in log entries, implicitly records the superseded Update. This way, when a parent communicates log entries to its child, it can refer to superseded Updates as "null" operations with sequence numbers, so that sequence numbers still advance contiguously, as a child expects.

5.1 Methodology

We implemented LOUP, iBGP with RRs and a generic flooding protocol that we will call BST* in a purpose-build event-driven network simulator[3]. We also have a real-world Quagga-based LOUP implementation that we will compare to Quagga's BGP [15].

We have explored LOUP's behavior on a wide range of synthetic topologies, including grids, cliques, and trees. These scenarios included varying degrees of link and router failure and the presence of MED attributes. In all cases LOUP's mechanisms worked as expected, requiring no explicit configuration and converging to the same solution as full-mesh iBGP.

To illustrate LOUP's behavior in a realistic scenario, we simulate a network based on that of Hurricane Electric (HE), an ISP with an international network. We use publicly available data: HE's complete topology including core router locations and iBGP data that reveals all next hops in the backbone [1]. These next hops are the addresses of either customer routers or customer-facing routers (nexthop-self). We assume there is an attachment point in the geographically closest POP to each distinct next hop, create a router for each attachment point and assign it to the closest backbone router. For iBGP-RR, we place RRs on the core routers and fully-mesh them. Recent studies suggest this model is not unrealistic [4].

We explore two different levels of redundancy. In the baseline redundancy case all clients in a POP connect to two aggregation routers, which in turn connect to the core router. In the more redundant case each aggregation router is additionally connected to the nearest POP. Unless explicitly specified all simulation results are from the more connected case.

We model speed-of-light propagation delay and add a uniform random processing delay in [0, 10] ms. We do not however model queues of updates that might form in practice, so our simulations should produce shorter-lived transients than might be seen in real backbones.

5.2 Correctness

To examine transient loops we compare the behavior of LOUP, BST*, and iBGP in two scenarios involving a single prefix: the announcement of a new "best" route for a prefix, and the withdrawal of one route when two routes tie-break on IGP distance. We compare both the less redundant and the more redundant topologies to observe the effect of increased connectivity. Figures 6 and 8 show the protocols' behavior when a single BR propagates an update, and all routers prefer that update to a route they are already using for the same prefix. As a result, this update triggers a withdrawal for the old route. And Figures 7 and 9 show the protocols' behavior in the tie-break withdrawal case.

We are interested in how the prefix's path from each router evolves over time. Define the correct BR before the change occurs as the old exit and the correct BR after the change occurs and routing converges as the new exit. In these four figures, we introduce the initial change at time $t = 0.1$ seconds and every 100 μs we check every router's path to the destination. Either a packet correctly reaches the new BR, still reaches the old BR, is dropped, or encounters a loop. The y-axis shows the number of routers whose path has each outcome. We plot the mean of 100 such experiments, each with randomly chosen BRs as the old and new exits.

Figures 6 and 8 confirm that LOUP incurs no transient loops or black holes and its convergence time is similar to that of the other protocols. BST* and iBGP-RR perform as expected; BST* does not cause black holes, but IBGP-RR causes both loops and black holes. On the less connected topology, there is limited opportunity for races to propagate far, so BST incurs relatively few loops. When it does loop, many paths are affected - the BST results have high variance. The more redundant the network, the more opportunity there is for BST to cause loops, as is evident from Figure 8.

Figs 7 and 9 demonstrate the importance of enforcing ordering on withdrawals. LOUP does not cause loops, but it takes longer to converge because the withdrawal first must propagate to the "tie" point and then be activated along the reverse path. All other protocols loop transiently because the BR immediately applies the withdrawal resulting in a loop like that in Fig. 4.

5.3 Scalability

To what extent do the different protocols concentrate processing load in a few routers? We take a set of 1000 routes from HE's iBGP data set, taking care to preserve all alternatives for each prefix we select, and inject them rapidly into the simulated HE network consisting of about 3700 routers. We rank the routers in terms of the messages sent or received, showing only the 450 busiest. The results are presented in Figs 10 and 13. In addition Figs 11 and 14 isolate only the BRs and demonstrate what the load would be in a BGP-free core environment.

Due to its flooding nature BST incurs a significant cost in all scenarios. LOUP is more expensive than iBGP, but not by a big margin. This is mostly due to the fact that route reflectors are able to shield their clients from some updates, unlike BST* and LOUP.

Running a large-scale routing protocol on commodity hardware is likely to be very different from running it on high-end routing platforms[6]. In particular FIB mod-

[3]We wanted to implement BST, but there is no clear spec, so it probably differs from BST in some respects.

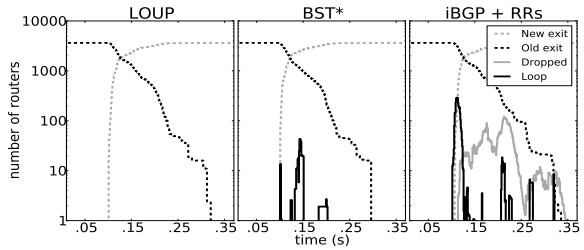

Figure 6: Transients on update (less connected)

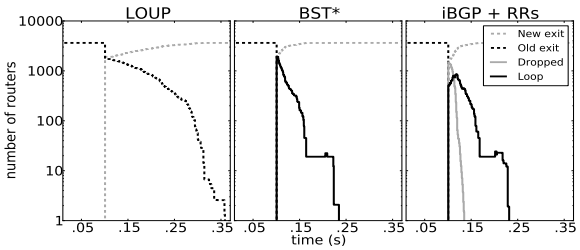

Figure 7: Transients on withdrawal (less connected)

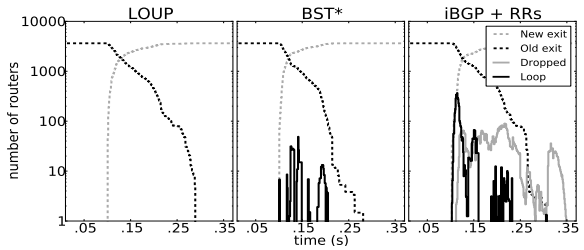

Figure 8: Transients on update (more connected)

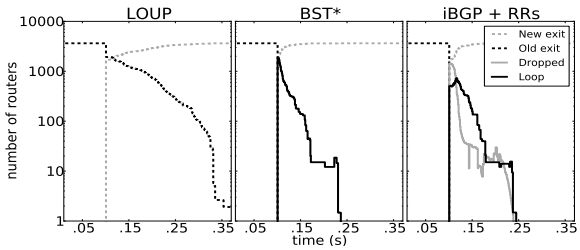

Figure 9: Transients on withdrawal (more connected)

ifications may be more expensive because they need to propagate from the control plane to the line cards. Generally FIB updates where an entry for a prefix already exists in the routing table are less costly than FIB adds or deletes where the trie may need to be rebalanced.

Figs 12 and 15 show the distribution of FIB operations during the experiment above. Even though iBGP processes fewer messages, those messages are more likely to cause FIB updates, and much more likely to cause expensive FIB adds or deletes. Route reflectors hide information - this may lead to path exploration during which the FIB is modified multiple times. LOUP and BST exhibit virtually identical behavior because they exchange the same information.

To evaluate CPU usage we run our Quagga-based LOUP implementation on commodity x86 hardware. We set up a simple topology, consisting of three single-core 2Ghz AMD machines (A, B and C) connected with two 1GBps links. In BGP's case we open an eBGP session from A to B and and iBGP session from B to C. In LOUP's case we perform no configuration. We inject one view of the global routing table (approx 400,000 routes) at A, which forwards it to B, which forwards it to C. We look at the load on the middle box (B), as it has to both receive and send updates and does the most work.

Task	LOUP	BGP
Updating the RIB	1981	5042
Updating the FIB	6544	16874
Serialization	3222	7477
Low-level IO	7223	6447
Other	2824	5369
Total (million cycles)	21797	41212
Total (seconds)	10.8	20.6

Both protocols spend most of the time updating the FIB and doing low-level IO. Running the decision process and updating the RIB datastructures is almost negligible. LOUP is much faster than BGP, but it seems that Quagga's BGP spends unnecessary time updating the FIB, and is not fundamental.

LOUP's memory usage, shown below, is directly dependent on the number of active alternatives that exist for a prefix (everything that does not break on IGP distance will be withdrawn).

BGP (1)	LOUP (1)	LOUP (2)	LOUP (3)
73.2 MB	46.7 MB	68.2 MB	89.8 MB

Memory usage is shown when we injected the same route feed from 1, 2 and 3 different BRs in our experimental network. We only present results for BGP with one view, because the RRs hide all but the winning routes from their clients. Because BGP has to maintain multiple RIBs for each session its memory footprint is higher than LOUP's. Based on HE's data, in a large ISP there will be on average 5-6 alternatives for a prefix. LOUP's memory usage grows linearly we expect the protocol to easily run on any modern-day platform in a network like HE's with around 200MB of RAM.

6 Related Work

There has been significant work on carefully disseminating routing updates so as to improve the stability of routing and ameliorate pathologies such as loops and black holes. We have discussed DUAL's approach to loop-free IGP routing [8], BST's reliable flooding approach to intra-AS route dissemination [16], and RCP's centralized approach to intra-AS route dissemination [3] at length in Sections 2 and 3. To recap: LOUP tackles loop-free intra-AS dissemination of externally learned routes, a different problem than loop-free IGP routing, as taken on by DUAL and oFIB [22]; the non-convexity of BST's

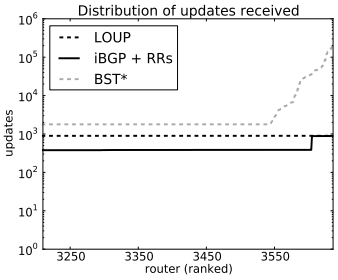

Figure 10: Updates received

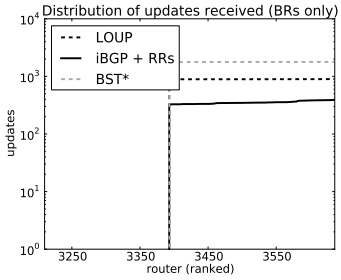

Figure 11: Updates received (BRs)

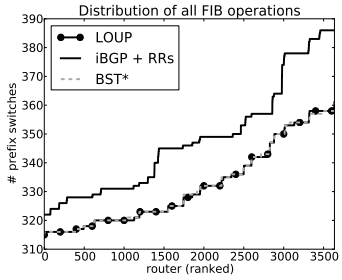

Figure 12: Total FIB changes

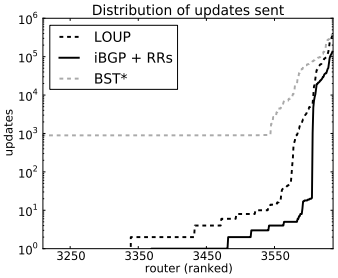

Figure 13: Updates sent

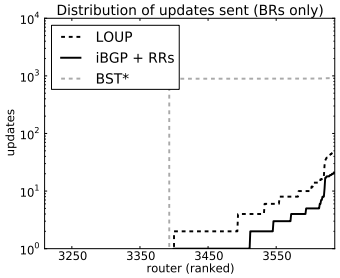

Figure 14: Updates sent (BRs)

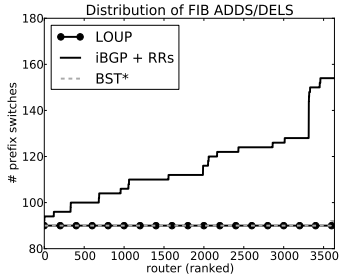

Figure 15: FIB adds and deletes

flooding causes transient loops that LOUP avoids; and RCP centralizes the BGP decision process for an AS, but does not propagate the results synchronously to all routers, and so does not achieve the freedom from transient loops and black holes that LOUP does. We note that loop-free IGPs like DUAL (and its implementation in EIGRP) complement LOUP nicely: running LOUP *atop* DUAL would prevent both IGP loops and transient loops present in today's route dissemination by iBGP with RRs. DUAL's "query" messages served as the inspiration for LOUP's "tell-me-when" mechanism in the "chatty" protocol described in Section 3.2.4.

Consensus Routing [17], briefly discussed in Section 1, adds Paxos-based agreement to eBGP to avoid using a route derived from an update before that update has propagated to some ASes. LOUP's combination of ordered, reliable dissemination of updates along an RFT with update logs is significantly lighter-weight than Paxos-based agreement, yet still avoids introducing loops within an AS during dissemination of an update or withdrawal. Bayou's logs of sequence-number-ordered updates [24] and ordered update dissemination [21] inspired the analogous techniques in LOUP; we show how to apply these structures to achieve robust route dissemination, rather than weakly consistent storage.

In our own prior work [13] (to appear in October 2012), we first proposed ordered, RFT-based dissemination as a means to avoid transient loops and black holes. In this paper, we have additionally described a full routing protocol built around these principles, and evaluated the scalability of a full implementation of that protocol atop the Quagga open-source routing platform.

7 Conclusion

The prevalence of real-time traffic on today's Internet demands greater end-to-end path reliability than ever before. The vagaries of iBGP with route reflectors—transient routing loops and black holes, route instability, and a brittle, error-prone reliance on configuration—have sent network operators running into the arms of MPLS, in an attempt to banish iBGP and its ills from the core of their networks. By exploring the fundamental dynamics of route dissemination, we have articulated why iBGP with route reflectors (and to an extent, alternatives like BST) introduce such pathologies. Based on these fundamentals, we have described a simple technique, ordered dissemination of updates along a reverse forwarding tree, that avoids them. And we have illustrated how to apply this technique in a practical, scalable routing protocol, LOUP, seen through to a prototype implementation. While earlier work has drawn upon consistency techniques from the distributed systems community to improve the robustness of routing, LOUP achieves strong robustness with lighter-weight mechanisms. As such, we believe LOUP offers a compelling alternative to a BGP-free core.

References

[1] Hurricane electric's looking glass server. `route-server.he.net`, July 2012.

[2] L. Andersson, I. Minei, and B. Thomas. LDP specification. *RFC 5036*, Oct. 2007.

[3] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and implementation of a routing control platform. In *Proc. ACM/USENIX NSDI*, 2005.

[4] J. Choi, J. H. Park, P. chun Cheng, D. Kim, and L. Zhang. Understanding BGP next-hop diversity. *IEEE INFOCOM*, Apr. 2011.

[5] L. De Ghein. *MPLS Fundamentals*. Cisco Press, 2007.

[6] K. Fall, G. Iannaccone, S. Ratnasamy, and P. Godfrey. Routing tables: Is smaller really much better? *Proc. ACM HotNets*, 2009.

[7] L. Gao, T. Griffin, and J. Rexford. Inherently safe backup routing with bgp. *IEEE INFOCOM*, 2001.

[8] J. Garcia-Luna-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking*, 1(1), Feb. 1993.

[9] T. Griffin and G. Wilfong. An analysis of BGP convergence properties. *SIGCOMM*, 1999.

[10] T. Griffin and G. Wilfong. Analysis of the med oscillation problem in bgp. *ICNP*, 2002.

[11] T. Griffin and G. Wilfong. On the correctness of IBGP configuration. In *Proc. SIGCOMM 2002*, Aug. 2002.

[12] T. Griffin and G. T. Wilfong. Analysis of the MED oscillation problem in BGP. In *Proc. ICNP '02*, pages 90–99, Washington, DC, USA, 2002.

[13] N. Gvozdiev, B. Karp, and M. Handley. LOUP: who's afraid of the big bad loop? *Proc. ACM HotNets*, Oct. 2012.

[14] IOS Hints Blog. Poll: Do you use MPLS for your internet traffic? `http://polldaddy.com/poll/2912704/`, 2010.

[15] K. Ishiguro et al. Quagga, a routing software package for TCP/IP networks. `http://www.quagga.net`, Mar. 2011.

[16] V. Jacobson, C. Alaettinoglu, and K. Poduri. BST - BGP scalable transport. *Presentation at NANOG 27*, Feb. 2003.

[17] J. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataraman. Consensus routing: The internet as a distributed system. In *Proc. NSDI 2008*, Apr. 2008.

[18] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?! It must be BGP. *ACM CCR*, Apr. 2007.

[19] D. McPherson, V. Gill, and D. Walton. Border gateway protocol (BGP) persistent route oscillation condition. *RFC 3345*, Aug. 2002.

[20] J. H. Park, R. Oliveira, S. Amante, D. McPherson, and L. Zhang. BGP route reflection revisited. *IEEE Communications Magazine*, July 2012.

[21] K. Petersen, M. J. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible update propagation for weakly consistent replication. In *Proc. SOSP 1997*, Oct. 1997.

[22] M. Shand, S. Bryant, S. Previdi, C. Filsfils, P. Francois, and O. Bonaventure. Loop-free convergence using ofib. *IETF Internet Draft draft-ietf-rtgwg-ordered-fib-06*, June 2012.

[23] V. Srinivasan and V. G. Fast address lookups using controlled prefix expansion. *ACM Transactions on Computer Systems*, 1999.

[24] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc SOSP 1995*, Dec. 1995.