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

Nikola Gvozdiev, Brad Karp, Mark Handley
University College London

Abstract

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 Simple Ordered Update Protocol (SOUP) and Link-Ordered Update Protocol (LOUP), clean-slate dissemination protocols for external routes that do not create transient loops, make stable route choices in the presence of failures, and achieve policy-compliant routing without any configuration. We prove SOUP cannot loop, and demonstrate both protocols' scalability and correctness in simulation and through measurements of a Quagga-based implementation.

1 Introduction

Much has been written about iBGP's susceptibility to persistent routing instability and forwarding loops [11, 12, 20]. Yet the transient behavior of intra-domain dissemination of external routes has been, to our knowledge, unexamined. In recent work, we found that today's iBGP frequently incurs transient forwarding loops while propagating updates [13]. Real-time traffic of the sort prevalent on today's Internet does not tolerate 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 any associated pathologies: border routers (BRs) must still use iBGP internally to exchange routes.

In this paper, we illustrate that the routing instability and transient loops 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 route dissemination protocols—not only iBGP as typically deployed with Route Reflectors (RRs), but in the case of transient loops, alternatives such as BST and RCP, as well—happen to disseminate routes. It is the lack of attention to the *order* in which routers adopt routes that causes these pathologies.

Based on these observations, we introduce the Simple Ordered Update Protocol (SOUP), a route dissemination protocol that *provably* never causes transient forwarding loops while propagating any sequence of updates from any set of BRs. SOUP avoids causing loops by reliably disseminating updates hop-by-hop along the *reverse forwarding tree* from a BR. We further introduce the Link-Ordered Update Protocol (LOUP), which uses a similar ordering mechanism to avoid loops, but includes optimizations that reduce its convergence time (compared with that of SOUP) in common cases.

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. SOUP and LOUP use 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 any set of updates to prefixes is introduced
- SOUP, a simple route dissemination protocol that enforces these invariants using ordered dissemination of log entries along a tree
- a proof that SOUP cannot introduce forwarding loops
- LOUP, an optimized route dissemination protocol that converges faster than SOUP
- 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 consists of three components: *External BGP* (eBGP) distributes routes between routing domains and is the instrument of policy routing. An *Interior Gateway Protocol* (IGP) such as OSPF or IS-IS tracks 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 concerned with iBGP: when a route changes elsewhere 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.

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 next hop—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 other routers in the domain so they can reach this destination. If a BR already has a route to that prefix, it only sends the new route to the other routers if it prefers the new route. Similarly, if a BR hears a route from another BR that it prefers 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.

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.

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 his 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 round 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.

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.

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

[1] ISPs often use two overlaid trees for redundancy.

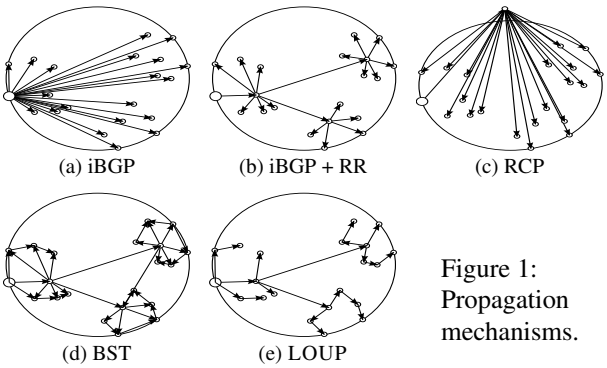

Figure 1: Propagation mechanisms.

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 from each entry BR to each exit BR.

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]. Another potential advantage of a BGP-free core is that it reduces the number of clients of iBGP route reflectors. Fewer clients mean less processing load on the RRs and less chance of configuration errors that cause routing instability.

Against these potential benefits, a BGP-free core depends on additional protocols such as LDP or RSVP-TE for basic connectivity, which add complexity and themselves need careful configuration. Moreover, a BGP-free core doesn't eliminate transient forwarding loops. BRs still run iBGP with RRs, and RRs still fail to control the order in which their distinct clients hear updates. 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 adoption of 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 describe replacements for iBGP that:

- are not susceptible to configuration errors,
- are stable under all configurations,
- are not prone to routing protocol traffic hot spots,
- minimize forwarding table (FIB) churn, both in BRs and internal routers,
- propagate no more changes to eBGP peers than full-mesh iBGP would,
- free of transient loops.

3 Dissemination Mechanisms

We now examine alternative route dissemination mechanisms from first principles to cast light on how forwarding loops arise and inform the design of the loop-free route dissemination protocol we describe subsequently.

Although iBGP's use of TCP ensures the in-order arrival of updates at each recipient, each recipient applies every update as soon as it can. Thus the order of update application *among* recipients is arbitrary, 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 constraint is insufficient to prevent loops and black holes.

One way to avoid both loops and persistent oscillations would be to centralize routing, as shown in Fig. 1c. When an external update arrives the BR first sends it to a routing control platform (*e.g.,* 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 controller has full knowledge, it could be extended to avoid loops by applying updates in a carefully controlled order. However, to do so would require a synchronous update approach which, given the RTTs to each router, would 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 next hop, it must already have applied the update, and so must be closer to the destination. Therefore 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. This constraint is DUAL's *feasibility condition,* and these routes are known as *feasible routes.* They are safe because no matter the router's distance after the update, it 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* [5] to make its choice. It *freezes* the current choice of next hop 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

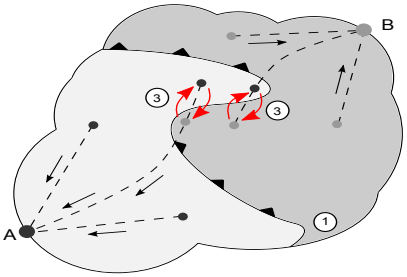

Figure 2: Transient loop when flooding an update.

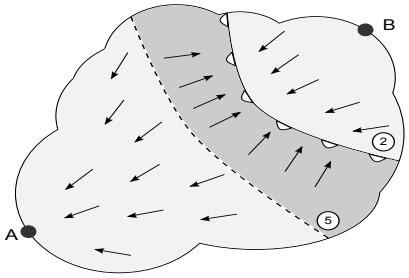

Figure 3: Withdrawal causes loop when alternate exists.

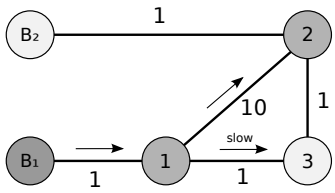

Figure 4: A simple topology exhibits looping.

choice. The responses spread out back across the network, activating routes in a wave as they return.

The iBGP route dissemination problem is different from that solved by DUAL, as iBGP distributes the same information to all internal routers, irrespective of the internal topology. DUAL, in contrast, concerns itself with path choice across the internal topology. For each prefix iBGP 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.

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, as shown in Fig. 1d, in which each router sends the messages it receives to all neighbors. Flooding must be done over one-hop reliable sessions to ensure messages are not lost. BST [15] takes this approach. Flooding imposes a topological ordering constraint, guaranteeing that at all times, a contiguous region of routers has processed an update. Essentially an update propagates across the domain as a *wavefront*; this is a necessary (though not sufficient) condition to avoid transient loops. iBGP does not have this property.

To see why this condition is not sufficient, even as a new route is propagated, 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 wavefront ① flowing outward 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 wavefront remains convex—that a forwarding path only crosses the wavefront once. As a result transient loops ③ can occur.

Fig. 4 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 normally be 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 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. Note that the update may be delayed either by the network (*e.g.,* congestion, packet loss) or more likely, by variable update processing delays at routers.

4 Simple Ordered Update Protocol

Building upon the discussion of route dissemination primitives above, we now propose two novel dissemination techniques, *reverse forwarding tree dissemination* and *backward activation.* We combine these into a Simple Ordered Update Protocol (SOUP), and prove that it never causes transient forwarding loops within an AS.

4.1 Reverse Forwarding Tree Dissemination

Recall that BST's loops occur when one BR propagates a new route that is preferred to a pre-existing route from another BR. A sufficient condition for avoiding such loops is for *no router to adopt the new route until the next hop for that route has also adopted the route.* The 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 in BGP route dissemination is for a router only to announce a route to routers that will forward via itself. Thus, route announcements flow from a BR along the reverse of the forwarding tree that packets take to reach that BR. Applying this condition in Fig. 4 precludes sending the update over link 1-2 as it is not on the RFT.

SOUP works by propagating announcements over a hop-by-hop tree, as shown in Fig. 1e. Unlike the Route Reflector tree, SOUP uses one tree per BR, rooted at that BR. SOUP builds this tree dynamically hop by hop by reversing the links on the shortest-path tree that the IGP follows to reach that BR from everywhere in the domain. This hop-by-hop nature preserves the wavefront property. Disseminating routes down the reverse forwarding tree (RFT) adds additional desirable ordering constraints that eliminate transient loops of the sort described above when improved routes are disseminated.

4.2 Sending Bad News: Backward Activation

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. This often occurs 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; some choose one exit, some another. Such hot-potato routing is common when two ISPs peer in multiple places.

Withdrawing one of those routes, as *B* does in Fig. 3, causes a loop. The routers behind withdrawal wavefront ② have already switched to an alternative route via *A*. Routers farther away have not yet heard the withdrawal and still forward to *B*. Traffic loops at wavefront ②.

To avoid such loops, when a BR withdraws a route, the change must first be applied by routers furthest from the BR. Essentially we want the change to propagate in exactly the reverse of what would happen when good news propagates. In this way, no router uses the to-be-withdrawn route to forward packets to a router that has already withdrawn the route, and so the withdrawal will not cause a loop to occur. In Fig. 3 the routers farthest from *B* will remove the route first, though doing so doesn't change their forwarding decision. The routers just to the right of ② will be the first to withdraw the route and change their choice of exit router, then their parents (with respect to the tree routed at B), and so on back up to B. We call this process *backward activation*.

4.3 The SOUP Protocol

SOUP routers actively build an RFT for each BR by exchanging messages with the relevant parent router. We describe this in more detail in Section 6.1. BRs receive route updates[2] over eBGP. If a BR uses a route or the update is a change to a route it previously used, then the BR sends it hop-by-hop down the RFT.

We define a route as *active* if it is eligible to be considered in the decision process, even if it is not installed in the FIB. Updates can be sent *forward activated* or *backward activated.* Each router makes its own choice of forward or backward activation, but with one exception, once the BR has originated a route as one or the other, the update will stay that way across the network.

When a router receives an update, it checks if the route is preferred to the current activated version of the same route it received from the same BR. If it is preferred, the route is *feasible;* it is applied immediately, and previous versions of this route from the same BR are flushed. If the route wins the BGP decision process, it is installed in the FIB. Irrespective of whether it was installed in the FIB, the router then sends it on to its children as forward activated. The children will also find it to be feasible.

If the route is not feasible, the router cannot yet apply the change. Its children still have the better version of this route, and if it applies the change, it may forward to a child who forwards right back again. Instead it keeps the old route active, adds the update to a list of inactive alternative routes received from that BR, and sends the change to all its children marked as *backward activated*.

When a backward activated update is received by a leaf router on the BR's RFT, it is safe for that router to activate the update—it has no children, so no loop can result. The leaf sends an *activation message* back to its parent, indicating that it has activated the change. Once the parent receives activation messages from all of its children, it in turn can activate the update and send its own activation message on to its parent. After activating an update, the router runs the BGP decision process in the normal way to decide which of its active routes, received from different BRs, it should install in its FIB.

SOUP's behavior is simple so long as only one update from a BR propagates at a time: good news forward activates and bad news backward activates. At no time does the existence or absence of an alternative route received from another BR change this dissemination process. But what happens when more than one change propagates simultaneously from the same BR?

Suppose a BR has already announced route ρ_1, then receives a route change from eBGP indicating the route got worse, becoming ρ_2. It sends an update containing ρ_2, which will backward activate. Before the activation messages have returned, the route improves somewhat, so the BR sends an update containing ρ_3, which will also backward activate—even though it is better than ρ_2, it is worse than the active route ρ_1. Each router therefore maintains a list containing $\{\rho_1, \rho_2, \rho_3\}$, where ρ_1 is still active, and ρ_2 and ρ_3 are awaiting backward activation.

At the leaves of the tree, ρ_2 has now activated, and the activation for ρ_2 spreads back up the tree. Descendants on the RFT below the backward activation only have ρ_2 in their list. At some point the update for ρ_3 propagating away from the BR passes the activation message for ρ_2 returning toward the BR. On the link where these messages cross, the child router, R_c receives update ρ_3. ρ_3's update is marked as *backward activated*, but is preferable to the active route ρ_2. It is therefore feasible, is applied immediately, and is sent on marked as *forward activated.* However, as ρ_3 was received at R_c marked as *backward activated*, R_c must also indicate to its parent that it has activated the route, so it sends an activation message.

These rules ensure that an update sent as forward activated is always propagated as forward activated, but an

[2] A withdrawal is just an extreme form of a route becoming less preferred, so we will only refer to updates from now on.

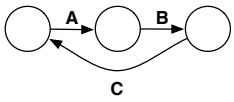

Figure 5: A simple routing loop.

update sent as backward activated may switch to forward activated if it crosses into a region where backward activation has already removed an older version of that same route. An update can only change from backward activated to forward activated once, and never the other way.

Limitations of SOUP SOUP is loop-free when disseminating changes if the IGP is loop free and internal routes do not change. It does not attempt to be loop-free while the IGP is reconverging, but as external route changes greatly outnumber internal ones in most networks, this seems a reasonable compromise.

SOUP cannot always prevent transient loops in the presence of the BGP MED attribute, though it will always converge to the same stable loop-free solution as full-mesh iBGP. With MED, although routes can be ranked as better or worse when originating from the same BR, the same is no longer true when they originate from different BRs. MED has the property that with three routes, *A*, *B*, and *C* for the same prefix, where three routers have routes *ABC*, *BC*, and *AC*, as might happen when routes *A* and *B* are propagating, then the three routers can choose routes *A*, *B*, and *C* respectively. For example, *B* beats *C* on MED, *A* beats *B* on router ID, *C* beats *A* on router ID. There is no total order among the three routes that all routers agree on, so looping can occur. iBGP-RR can exhibit persistent route oscillations in this case, and Cisco provides an *always-compare-MED* option to impose a strict total order. We conjecture that no distributed protocol can ensure loop-freedom when MED is used without such a workaround.

4.4 Proof of SOUP's Loop Freedom

To show that SOUP does not introduce forwarding loops, we focus on how the protocol handles updates for a single prefix. Our proof rests on two assumptions: first, that the IGP is not currently reconverging; and second, that BGP's always-compare-MED option is in use (such that there is a strict total order on BGP routes). Without these assumptions, SOUP cannot always prevent transient loops. We proceed in two steps: first, we introduce a sufficient condition for loop avoidance, and second, we prove that SOUP always complies with that condition.

Lemma. *When the quality of the route used for forwarding improves monotonically at all successive router hops, no forwarding loop can occur.*[3]

Proof. Consider the simple three-router topology in Fig. 5, in which letters denote the routes on which each router forwards. Define the operator "$\prec$" such that for routes x and y, $x \prec y$ means that the BGP decision process prefers y to x. Consider a path whose first edge (route) is A, along which route quality monotonically increases. We proceed by contradiction. Assume that a loop occurs along this path, and without loss of generality, assume the loop occurs between the third and first router, as shown in Fig. 5. But if this loop occurs, we have that $A \prec C$ (by the path's monotonic improvement) and also that $C \prec A$, as the loop will forward successively on routes C and A. In general, for any path on which routes improve monotonically, a forwarding loop will cause a contradiction in which the route that closes the cycle must simultaneously be less preferred and more preferred than the immediately subsequent route. □

Theorem. *SOUP does not introduce forwarding loops.*

Proof. When a packet is forwarded, the routers along the path do not have to forward towards the same BR. Some may have received new state that the others have not yet heard. However, as shown in the above lemma, so long as the routing state along the path monotonically improves, no loop can occur. For a loop to occur, monotonicity must be violated: somewhere, a router r_{n+1} must forward towards BR r_0, and the next hop, r_n, must forward using less good state than that used by r_{n+1}. Let the route being used at r_n be ρ_n* and the route being used at r_{n+1} be $\rho_{n+1}*$. (We will use $*$ to indicate that a route is installed in the FIB and used for forwarding.) To violate monotonicity, and thus for a loop to be possible, $\rho_n* \prec \rho_{n+1}*$. There are two cases to consider, depending on which BR originated ρ_n*:

Case 1: $\boxed{\rho_n* \text{ is a route that originated at } r_0.}$

For r_{n+1} to have route $\rho_{n+1}*$, it must previously have received a forward activated update for a route ρ_{n+1}^{best} from r_0 that was either better than $\rho_{n+1}*$ or is actually $\rho_{n+1}*$. This is the case because forward activated updates are the only way routes can improve. As r_{n+1} is the child of r_n with respect to BR r_0, for r_{n+1} to have received ρ_{n+1}^{best}, it must have received this route from r_n. r_n must therefore have also held ρ_{n+1}^{best} at some point.

If $\rho_n* \prec \rho_{n+1}*$, then r_n must have received an update that replaced ρ_{n+1}^{best} with a worse route. Such an update must have backward activated at r_n, as it would not have been feasible compared to ρ_{n+1}^{best}. However, for ρ_n* to have been backward activated, as r_{n+1} is the child of r_n, the activation must have passed through r_{n+1}, and it would have replaced $\rho_{n+1}*$. Thus r_{n+1} cannot have $\rho_{n+1}* \succ \rho_n*$ if ρ_n* originated at r_0.

To summarize: if r_{n+1} has route ρ_{n+1}, then its parent r_n must still have the same route or a better one.

Case 2: $\boxed{\rho_n* \text{ originated at BR } r_{BR}, \text{ where } r_{BR} \neq r_0.}$

However, by case 1, if r_{n+1} has route $\rho_{n+1}*$ received from r_0, then r_n must still have a route $\rho_n^{r_0}$ received

[3] Jaffe and Moss offer a similar proof [16].

from r_0, such that $\rho_{n+1}* \prec \rho_n^{r_0}$. If router r_n has route $\rho_n^{r_0} \succ \rho_n*$, then it will not choose ρ_n*—the BGP decision process can only choose $\rho_n* \succ \rho_n^{r_0}$. Hence it cannot be the case that $\rho_n* \prec \rho_{n+1}*$.

As neither case 1 nor case 2 can occur, it is impossible for a loop to occur. □

5 Link-Ordered Update Protocol

SOUP avoids loops by always maintaining the invariant that no router ever has better active state for a prefix from a specific BR than its parent router. The down side of maintaining this invariant is that bad news must normally propagate all the way across the network and the backward activations return before worsening or withdrawn state can be removed. If, after receiving bad news from eBGP, a BR still has an external route, even though it is no longer the best route from the domain, then there is no significant problem: the destination is still reachable, and all that happens is a suboptimal path is used for a short while. However, when the BR receives a withdrawal via eBGP and has no other eBGP route, then it will have to drop any packets for this destination. SOUP forces the BR to hold the old route longer than we would wish, prolonging the black hole longer than alternative protocols that are not loop-free.

Is it possible to get the best of both worlds: maintain loop-free routing, but also reconverge quickly? Backward activation of withdrawals is essential for loop-free routing, but inevitably delays switching to an alternative route. Thus SOUP cannot converge as fast as protocols that do not perform backward activation. However, it is often safe to terminate a backward activation without having it traverse the whole network and back.

Consider Fig. 3, where the route advertised by B is being withdrawn and an alternative route from A that tie-broke on IGP distance exists. To avoid loops, SOUP's withdrawal wavefront spreads the whole way across the network before activating on the reverse path. It is safe instead to activate the withdrawal as soon as the wavefront reaches ⑤. The first hop to the left of ⑤ could trigger the backward activation of the withdrawal by sending a reply. This short-cutting of backward activation is the essence of the Link-Ordered Update Protocol (LOUP).

5.1 Local Activation of Withdrawals

Just as with SOUP, a LOUP router passes a worsening update on to its children without activating it, and waits for backward activations from them before activating the route change. Doing so maintains the invariant that a router never has a better activated version of the route than its parent does. This invariant ensures that successive announcements of a route from the same BR cannot cause a loop.

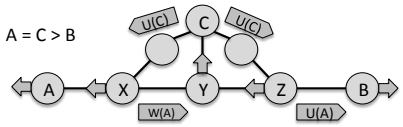

Figure 6: Local activation can lead to transient loops.

Fig. 3 might lead us to consider that the routers just to the left of ⑤ could locally activate the withdrawal. They are not using the route being withdrawn, so they could reply immediately to their parent, activating the withdrawal back toward B. In the steady state, doing so is safe and does not cause loops. However, it violates the invariant that no router ever has a better activated version of a route than its parent. Although it is quite difficult to find cases where a loop results, they do exist.

Consider Fig. 6. The three BRs, A, B, and C have all announced routes, such that $A = C \succ B$. The updates from A and C have not yet reached B, so B has not yet withdrawn its route. C's update has reached Y, but not reached X and Z. A has withdrawn its route—the withdrawal will backward activate. Thus A, X, and Z are currently using route A (shown by the block arrows), Y is using C, and B is using its own route. So far, the network is loop-free.

The withdrawal of A then reaches Y, which is not using route A. Y therefore activates the withdrawal, passes it to its children, and sends an activation message back to X. Before any of these messages are received, Y then receives a withdrawal of C. Because neither X nor Z reaches C via Y, Y is a leaf on C's RFT. Y immediately activates the withdrawal, and switches to its only alternative, which is B. Traffic now loops between Y and Z.

Thus we can see that local activation is not safe in the presence of transient announcements such as the one from C. The problem is that the existence of the route from C causes routers Y and Z to violate the invariant—Z still has the route to A, but Y does not. To maintain the invariant and be loop-free, the routing state at a route for one BR must not depend on the existence or absence of routing state for another BR.

5.2 Targeted Tell-me-when

To avoid the problem associated with local activation, we build upon the sound basis of SOUP. LOUP activates an update exactly as SOUP does—it triggers backward activation under the same conditions as SOUP. However, if a router has an alternative route via another neighbor, it is safe to switch to that route if the router knows that neighbor is no longer using the route that is waiting for a backward activation.

Upon receiving a withdrawal (or a worsening update), a router marks the route as backward activated and passes it on to its children. It then also runs the decision process to determine which route it would use if the change

were activated. If there is an alternative route via another neighbor, the router sends a *tell-me-when* request to that neighbor requesting that it reply when it is no longer using the route being withdrawn.

If the response to the *tell-me-when* arrives before the activation, the router knows that it is now safe to use the alternative because its new next hop (and transitively, all routers between it and the new exit point) is already using the alternative. The router does not yet activate the withdrawal, but it can reply to any *tell-me-when* from its other neighbors.

If the alternative route is withdrawn, the router still has the original route (waiting for backward activation of the withdrawal) in its table, and can safely switch back to it if necessary as the invariant still holds, so long as it has not replied to any other router's *tell-me-when* for this route. If it has replied to a *tell-me-when*, the route is marked as *dead*. It is unsafe to switch back to a dead route, and packets for this prefix will be black-holed until the backward activation arrives.

Only when the backward activation arrives from all a router's children is a withdrawal finally activated.

5.3 Safety

With respect to a BR, LOUP maintains the same invariant as SOUP, so no two routers forwarding on state from the same BR can cause a loop. In [13], for an older version of LOUP that used targeted tell-me-when messages but did *not* use backward activation, we exhaustively considered all the possible ways a single update or withdrawal could interact with existing forwarding state at routers. So long as the IGP itself is loop-free, there was only one way a loop could occur: a race condition we called a Withdrawal/Announcement race, where a withdrawal of one route caused a previously suppressed route to be reannounced. The triggering withdrawal and the triggered announcement could race, leading to loops. The current LOUP protocol's backward activation mechanism prevents this race condition. We thus assert that no single update can cause LOUP to create a forwarding loop. We make no assertion that LOUP is loop-free when a BR sends multiple updates for the same prefix in extremely rapid succession, but we have not seen such loops in simulation. BGP's MRAI timer would normally prevent this.

5.4 Freedom from Configuration Errors

Full-mesh iBGP requires all peerings be configured. The configuration is simple, but all routers must be reconfigured whenever any 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. A BGP-free core improves iBGP's scaling somewhat, at the expense of requiring additional non-trivial mechanisms just to route traffic across the network core. All this configuration significantly increases the likelihood of disruptions caused by configuration errors.

Hop-by-hop dissemination mechanisms such as BST and LOUP are configuration-free. All one must do is enable the protocol. Some might equate configuration with control. We will show that LOUP's freedom from configuration does not give rise to routing protocol traffic hotspots.

6 Building and Using the RFT

We now describe the details of ordered update dissemination along an RFT in the SOUP and LOUP protocols.[4] There are two main aspects: how to build the RFT, and how to disseminate updates along the RFT reliably despite topology changes.

6.1 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 RFTs. 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. Whenever the RFT changes, each LOUP router 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] Both protocols use the exact same RFT techniques; we write "LOUP" hereafter for brevity.

6.2 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 an active operation and one or more inactive 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. Inactive operations are backward propagated updates that have not yet been activated. 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.

LOUP requires that the topology within an AS remains stable long enough for LOUP to establish parent-child adjacencies with its Child messages. So long as this condition holds, LOUP's single-hop TCP connections coupled with its log mechanism guarantee reliable dissemination of operations down the RFT. Topology changes may temporarily disrupt the RFT, but all data will eventually reach the entire AS.

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 routers receive the operation, they apply it to their logs. When all operations are eventually activated 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.

7 EVALUATION

We evaluated SOUP and LOUP to examine their correctness, scalability, and convergence speed. To be correct they must:

- always converge to the same solution as full-mesh iBGP—doing so guarantees no persistent oscillations if eBGP policy obeys AR; and
- not create transient loops, so long as the underlying IGP's routes are loop-free.

We assess scalability by asking:

- How is the CPU load distributed between routers?
- How are FIB changes distributed? Is the FIB updated more frequently than with iBGP?
- How much churn is propagated to neighboring ASes?
- What is the actual cost of processing updates? Can bursts of updates be handled quickly enough?
- Can the implementation hold the global routing table in a reasonable memory footprint? Neither SOUP nor LOUP hides information, so how well does it compare to BGP with RRs?

Finally, we consider the delay and stability behavior of these protocols during convergence, and compare them with the alternate loop-free strategy of injecting external routes into DUAL:

- How do the convergence times of SOUP and LOUP compare? How long does DUAL take to converge when conveying external routes?
- Does injecting external routes into DUAL render the network unstable? What is the cost when an internal link comes up or down?

7.1 Methodology

We implemented LOUP, SOUP, iBGP with RRs and a generic flooding protocol that we will call BST* in a purpose-built event-driven network simulator.[5] We also implemented a version of DUAL that injects external

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

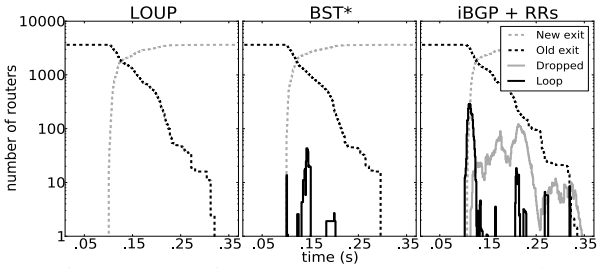

Figure 7: Transients on update (less connected)

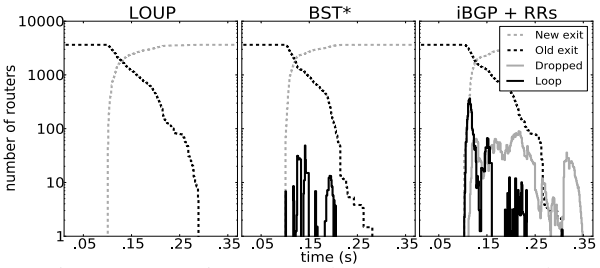

Figure 9: Transients on update (more connected)

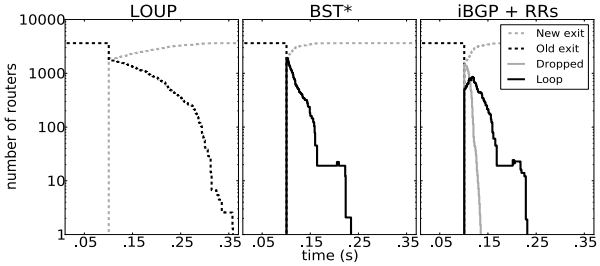

Figure 8: Transients on withdrawal (less connected)

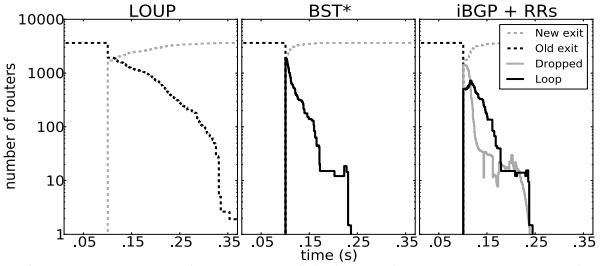

Figure 10: Transients on withdrawal (more connected)

routes, so it can be used as a loop-free iBGP replacement. In addition we implemented LOUP for Quagga [14], and compare it to Quagga's iBGP implementation.

We have explored the behavior of SOUP and LOUP 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 the two protocols required no explicit configuration and converged to the same solution as full-mesh iBGP.

To illustrate the behavior of the protocols 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 backbone topology (including core router locations) and iBGP data that reveal all next hops in the backbone [1]. These next hops are the addresses of either customer routers or customer-facing routers. 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 connect them in a full mesh. 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.

In the case of DUAL, we inject eBGP routes into DUAL by mapping the BGP path attribute list to a DUAL metric, and then allow DUAL to distribute these routes internally as it normally does.

7.2 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. Figs. 7 and 9 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 Figs. 8 and 10 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.

Figs. 7 and 9 confirm that LOUP incurs no transient loops or black holes and its convergence time is similar to

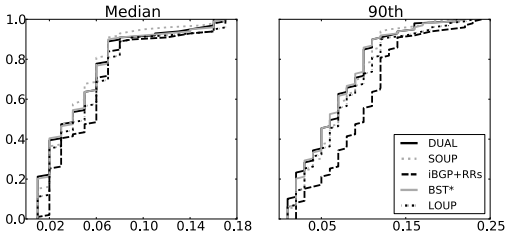

Figure 11: Convergence delay on update (sec)

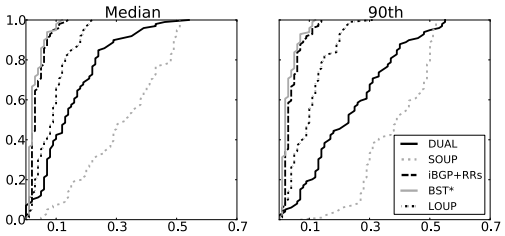

Figure 12: Convergence delay on withdrawal (sec)

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 Fig. 9.

Figs. 8 and 10 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. 3.

In the interest of brevity we omitted graphs for DUAL and SOUP, but neither incurred transient loops. We examine their convergence latency separately below.

7.3 Convergence Delay

How quickly do the various protocols propagate changes to all routers? We repeated the single-update and single-withdrawal experiments from Fig. 10 for 100 passes per protocol. We collected the median and 90th percentile delays of all passes and present their CDFs. We also present results for DUAL and SOUP. The results are in Figs. 11 and 12. When disseminating good news, all the protocols incur similar delay.

Fig. 12 shows the price paid to avoid looping with bad news. BST and iBGP converge fastest, but cause transient loops. SOUP performs worst, as it cannot short-cut the activation of withdrawals, propagating them all the way to the end of the AS, and only then activating. In fact, if one or two routers were slower, this effect would be exacerbated. Both LOUP and DUAL overcome this

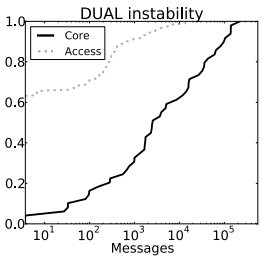

Figure 13: DUAL instability

problem, but DUAL's flooding nature means that it is a little slower to converge. When frozen, each DUAL router must collect replies from all its neighbors before terminating the diffusing computation, while a LOUP router only needs to wait for its children on the RFT. This is the reason for the gap between DUAL and LOUP.

7.4 Stability

The most significant problem with distributing external routes into DUAL is the increased cost of IGP changes. After LOUP, SOUP or BST has propagated external routes through the network, any route change in the IGP will not cause additional churn—the IGP will reconverge, but there will be no need to re-exchange external routing information. If a link fails with DUAL, a new path is calculated for each destination, including the external ones, as each may be injected from a different subset of BRs. To do this requires a great deal of communication. In some cases a single link's failure may cause all external routes to be frozen and DUAL will have to re-converge for each of them.

We injected 5000 routes from the HE data set into DUAL—these relate to 300 different prefixes. We then failed one link and observed the traffic generated that relates to these external prefixes. We repeated this process for every core link and 300 randomly chosen access links. Fig. 13 shows a CDF of the results. For 50% of the access links, a failure results in more than 2000 messages being exchanged and 10% of link failures generate more than 10000 messages. We would expect message complexity to scale linearly; for example, with a full routing table of 300K prefixes, we would expect 50% of core link failures to result in more than 2 million messages being sent.

Access link failures generate fewer messages—60% generate none in this experiment because these BRs injected no external routes. With a full routing table, all these would have injected some routes. Of the ones that were on the shortest-path tree for some external prefix, 50% of link failures generated more than 300 messages. With a full routing table, we would expect this number to grow to around 300K messages.

We have omitted results for LOUP, SOUP and BST because they generate no routing messages for exter-

nal prefixes when the IGP changes, though LOUP and SOUP generate up to one message per neighbor to maintain the RFTs. iBGP does generate some churn as RRs change preferred routes and inform their clients of the changes, but this churn is usually insignificant compared to DUAL's, and it does not manifest in this scenario.

7.5 Scalability

To what extent do the different protocols concentrate processing load in a few routers? We take a set of 10000 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 3000 router HE network. We rank the routers in terms of messages sent or received, and show the 750 busiest in Figs. 14, 15, and 16. Message counts are a measure of communication cost, and update counts are a measure of the cost of running the BGP decision process.

BST's flooding means it incurs greater communication cost and processes many updates. LOUP and SOUP are a little more expensive than iBGP, as route reflectors hide some updates from their clients. As the HE route set does not include many external withdrawals, tell-me-when kicks in rarely, so LOUP and SOUP perform almost identically. DUAL's overall costs are similar, but it performs more processing in well-connected core routers as it explores alternatives.

Control-plane overhead is only one aspect to scalability: on some routing hardware, FIB changes are the routing bottleneck [6]. Generally FIB adds or deletes where the trie may need to be rebalanced are more costly than in-place updates to existing entries.

Figs. 17 and 18 show FIB operations during the experiment above. All protocols except DUAL perform a similar number of operations. Although iBGP processes fewer messages, those messages are more likely to cause expensive FIB adds or deletes. Route reflectors hide information. Doing so can lead to path exploration during which the FIB is modified multiple times, but it may also shield RRs' clients from a number of FIB updates. SOUP, LOUP, and BST exhibit virtually identical behavior because they exchange the same information.

The DUAL results show the number of times DUAL changes successor. It can do so often, as its loop-avoidance mechanism needs to freeze and then unfreeze portions of the network when updates for the same prefix propagate at the same time.

Fig. 19 shows FIB operations at the BRs only. When a BR changes route, it usually notifies its external peers, so this is a measure of churn passed on to eBGP. DUAL is significantly worse than the other protocols here, as it explores more alternatives before converging.

To evaluate CPU usage we run our Quagga-based LOUP implementation using a simple topology consisting of three single-core 2Ghz AMD machines (*A*, *B* and *C*) connected with gigabit links. In BGP's case we open an eBGP session from *A* to *B* and an iBGP session from *B* to *C*. In LOUP's case we perform no configuration. We inject one view of the global routing table ($\tilde{4}$00,000 routes) at *A*, which forwards to *B*, which forwards to *C*. We look at the load on *B* as it must 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 data structures is almost negligible. LOUP is much faster than BGP, but it seems that Quagga's BGP spends unnecessary time updating the FIB, so this effect is not fundamental.

LOUP's memory usage, below, depends directly on the number of routes for a prefix that tie-break on IGP distance, as other alternatives 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 greater 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, so we expect the protocol to run easily in a network like HE's on any modern router with 200 MB or more of RAM.

8 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 [15], 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 flooding causes transient loops that LOUP avoids; and RCP centralizes the BGP decision process for an AS, but does

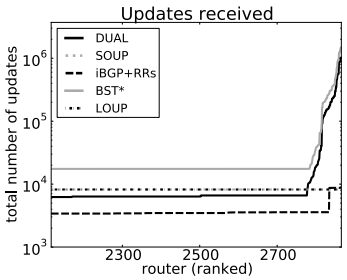

Figure 14: Updates received

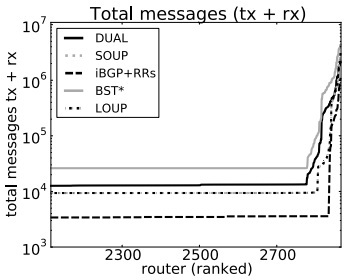

Figure 15: Total messages

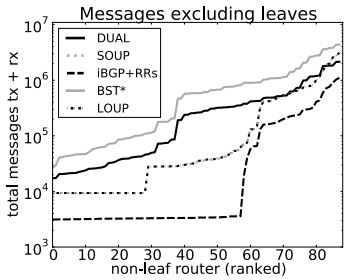

Figure 16: Messages (no leaves)

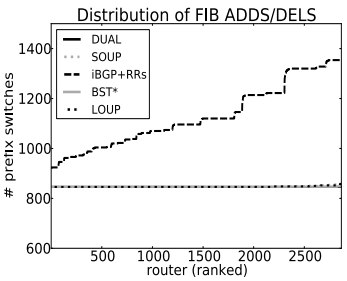

Figure 17: FIB adds and deletes

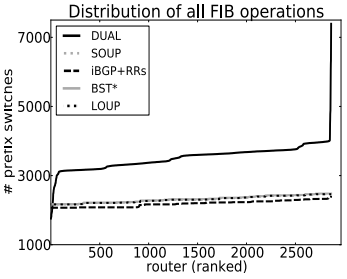

Figure 18: FIB operations

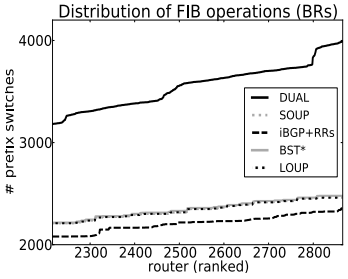

Figure 19: FIB operations (BRs)

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.

Consensus Routing [17] adds Paxos-based agreement to eBGP to avoid using a route derived from an update until that update has propagated to all ASes. LOUP's ordered, reliable dissemination of updates along an RFT is lighter-weight than Paxos-based agreement, yet still avoids introducing loops within an AS during dissemination. 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 prior work [13], we first proposed ordered, RFT-based dissemination as a means to avoid transient loops. In this paper, we have additionally described SOUP and LOUP, full routing protocols built around these principles, proven that SOUP never causes forwarding loops, and evaluated the scalability of a full implementation of LOUP atop the Quagga open-source routing platform.

9 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, 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. In exploring the fundamental dynamics of route dissemination, we have articulated why iBGP with route reflectors and 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 practice. SOUP is provably loop-free, but incurs latency associated with network-wide backward activation of less preferable routes. LOUP converges faster than SOUP by short-cutting backward activation in common cases. During convergence after a single update from a single BR, LOUP prevents forwarding loops. But as LOUP may incur loops under heavy update churn for the same prefix from multiple BRs, it trades absolute loop-freedom for faster convergence. Our evaluation in simulation has revealed LOUP to be a practical, scalable routing protocol, which we have also seen through to a prototype implementation for Quagga. While earlier work has drawn upon consistency techniques from the distributed systems community to improve the robustness of routing, SOUP and LOUP achieve strong robustness with lighter-weight mechanisms. As such, we believe they offer compelling alternatives to a BGP-free core.

Acknowledgements

We thank Arvind Krishnamurthy, our shepherd, and the anonymous referees for their insightful comments. This research was supported by EU FP7 grants 287581 (OpenLab) and 257422 (Change), and by a gift from Cisco.

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] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

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

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

[16] J. Jaffe and F. Moss. A responsive distributed routing algorithm for computer networks. *IEEE Transactions on Communications*, 30(7):1758–1762, 1982.

[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.