Consensus Routing: The Internet as a Distributed System

John P. John[*] *Ethan Katz-Bassett*[*] *Arvind Krishnamurthy*[*] *Thomas Anderson*[*]

Arun Venkataramani[†]

University of Washington Technical Report UW-CSE-08-02-01

Abstract

Internet routing protocols (BGP, OSPF, RIP) have traditionally favored responsiveness over consistency. A router applies a received update immediately to its forwarding table before propagating the update to other routers, including those that potentially depend upon the outcome of the update. Responsiveness comes at the cost of routing loops and blackholes—a router A thinks its route to a destination is via B but B disagrees. By favoring responsiveness (a liveness property) over consistency (a safety property), Internet routing has lost both.

Our position is that consistent state in a distributed system makes its behavior more predictable and securable. To this end, we present consensus routing, a consistency-first approach that cleanly separates safety and liveness using two logically distinct modes of packet delivery: a *stable* mode where a route is adopted only after all dependent routers have agreed upon it, and a *transient* mode that heuristically forwards the small fraction of packets that encounter failed links. Somewhat surprisingly, we find that consensus routing improves overall availability when used in conjunction with existing transient mode heuristics such as backup paths, deflections, or detouring. Experiments on the Internet's AS-level topology show that consensus routing eliminates nearly all transient disconnectivity in BGP.

1 Introduction

Internet routing, especially interdomain routing, has traditionally favored responsiveness, i.e., how quickly the network reacts to changes, over consistency, i.e., ensuring that packets traverse adopted routes. A router applies a received update immediately to its forwarding table before propagating the update to other routers, including those that potentially depend upon the outcome of the update. Responsiveness comes at the cost of availability: a router A thinks its route to a destination is via B but B disagrees, either because 1) B's old route to the destination is via A, causing loops, or 2) B does not have a current route to the destination, causing blackholes. BGP updates are known to cause up to 30% packet-loss for two minutes or more after a routing change, even though usable physical routes exist [25]. Further, transient loops account for 90% of all packet loss according to a Sprint network study [16]. Even a recovering link can cause unavailability lasting tens of seconds due to an inconsistent view among routers in a single autonomous system [43].

Our position is that the lack of consistency is at the root of bigger problems in Internet routing beyond availability. First, protocol behavior is complex and unpredictable as routers by design operate upon inconsistent distributed state, e.g., by forwarding packets along loops. There is no indicator of when, if at all, the network converges to a consistent state. Second, unpredictable behavior makes the system more vulnerable to misconfiguration or abuse, as it is difficult to distinguish between expected behavior and misbehavior. Third, unpredictable behavior stifles innovation in the long term, e.g., network operators are reluctant to adopt protocol optimizations such as interdomain traffic engineering [1] because they have to worry about its poorly understood side-effects. Perhaps most tellingly, despite a decade of research investigating the complex dynamics of interdomain routing, the *goal of a simple, practical routing protocol that allows general routing policies and achieves high availability* has remained elusive.

Our primary contribution, consensus routing, achieves the above goal. The key insight is to recognize consistency as a safety property and responsiveness as a liveness property and systematically separate the two design concerns, thereby borrowing an old lesson from distributed system design. Consistency safety means that a router forwards a packet strictly along the path adopted by the upstream routers unless the packet encounters a failed link. Liveness means that the system reacts quickly to failures or policy changes. Separating safety and liveness improves end-to-end availability, and, perhaps more importantly, makes system behavior simple to describe and understand.

Consensus routing achieves this separation using two logically distinct modes of packet delivery: 1) A *stable mode* ensures that a route is adopted only after all dependent routers have agreed upon a consistent view of global state. Every epoch, routers participate in a distributed snapshot and consensus protocol to determine whether or not updates are *complete*, i.e., they have been processed by every router that depends on the update. The output of the consensus serves as an explicit indicator that routers may adopt a consistent set of routes processed before the snapshot. 2) A *transient mode* ensures high availability when a packet encounters a router that

[*]Dept. of Computer Science, Univ. of Washington, Seattle.
[†]University of Massachusetts Amherst.

does not possess a stable route, either because the corresponding link failed or the consensus protocol to compute a stable route has not yet terminated. In this case, the router explicitly marks it as a transient packet, and uses local information about available routes to heuristically forward the packet to the destination. We show that consensus routing can cleanly accommodate a number of existing transient forwarding heuristics such as backup routes [23], deflections [46], and detours [45] to provide near-perfect availability in a policy-compliant manner.

Consensus routing is similar in spirit to recent work on intradomain routing protocols that advocate the separation of route computation from packet forwarding [26]. However, we address this challenge for interdomain routing where ASes can run *arbitrary and private policy* engines to select routes, thereby precluding the use of logically centralized schemes for route computation. Consensus routing needs no change to BGP, residing as a layer on top of existing BGP implementations. A consensus router logs the output of the BGP policy engine locally, and uses the global consensus algorithm only to determine the most recent consistent BGP state; thus, an AS does not disclose any more information about its preferences than with BGP. We believe the consensus routing design also applies to intradomain routing without the need for a central policy engine.

In summary, our primary contribution is a simple, practical routing protocol that allows general policies and achieves high availability. To this end, we present

1. Consensus routing, a policy routing protocol that systematically separates safety and liveness concerns using two modes: a stable mode to ensure consistency, and a transient mode to optimize availability.
2. Provable guarantees that packets traverse adopted loop-free routes under general policies.
3. Experimental results based on the current Internet graph that show that consensus routing achieves high availability amidst link failures and policy changes.
4. A proof-of-concept prototype of consensus routing based on XORP [19] showing that the proposed design is practical and incurs little processing overhead.

2 A case for consistency

We illustrate several examples where inconsistent forwarding tables cause transient unavailability in inter- and intra-domain routing. These examples are well known, and while some solutions have been proposed to address each, our contribution is a comprehensive but simple solution to the suite of problems. In each case, the unavailability could last several tens of seconds (and sometimes minutes) due to BGP message processing and propagation delays [25]. For an introduction to BGP, refer [41].

1. *BGP link failures*: Figure 1 shows how link failures cause transient loops in BGP. Bold lines show selected paths. If link 4-5 goes down, 4 would immediately send a withdrawal to 2 and 3. However, because both 2 and 3 know of alternate paths 3-4-5 and 2-4-5 respectively, they start to forward traffic to each other causing loops. The MRAI timer prevents 2 and 3 from advertising the new paths even though they have adopted them to forward traffic. The timer is believed necessary to prevent a super-exponential blowup in message overhead, and its recommended value is 30 seconds. Eventually, when the timer expires, both 2 and 3 discover the alternate path to 5 through 1 that existed all along.

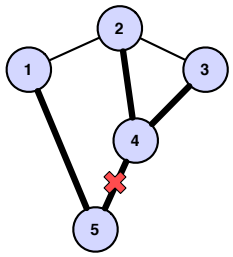

Figure 1: Link failure causing BGP loops at 2 and 3.

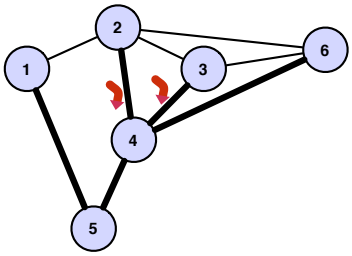

Figure 2: Policy change causing BGP loops at 2 and 3 when 4 withdraws a prefix from 2 and 3 but not 6.

2. *BGP policy change*: Figure 2 shows an example of how policy changes cause routing loops. AS4 may wish to engineer its traffic by withdrawing a prefix from 2 and 3 while continuing to advertise it to 6 for load balancing purposes [38]. (For instance, by diverting traffic to arrive from 6 instead of 2, internal congestion within AS4 might be decreased.) If 2 and 3 each prefer the other over 6, routing loops would result like in Figure 1. A similar situation also occurs if 5 wishes to switch its primary (backup) provider from 4 (1) to 1 (4); in this case, 5 is forced to either withdraw the route advertised (and potentially being used) to 4, or wait for a reliable indicator of when all traffic has completely moved over to the new primary provider 1. Other gadgets involving longer unavailability due to policy changes may be found in [30, 34].

3. *iBGP link recovery*: Figure 3 shows a transient blackhole caused by iBGP inconsistency. Routers A, B, and C belong to AS1 while D belongs to the adjacent AS2. iBGP is a BGP protocol that runs between routers inside an AS (in this case, A, B, and C). All routers route

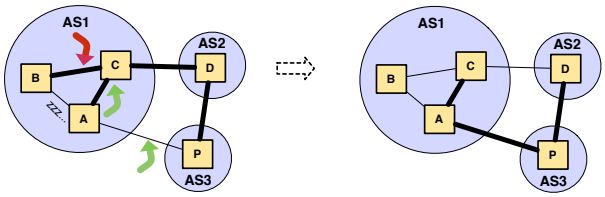

Figure 3: iBGP link recovery causing blackholes.

via D to the destination P in AS3. Suppose the previously failed link A-P recovers and is preferred by all AS1 routers over the route via AS2. If the AS1 routers all peer with each other, C will withdraw C-D-P from both A and B when it hears from A that AP is available, but will leave it to A to announce AP to B directly because of the full-mesh design. If A is waiting upon its iBGP timer, B experiences a transient blackhole. The current BGP spec recommends an iBGP timer shorter than interdomain timers, and typical values range from 5-10 seconds. Wang et al. [43] note that such blackholes can cause packet loss for tens of seconds. If AS1 routers use route reflection as opposed to full-mesh, similar consistency problems can cause unavailability [15, 3].

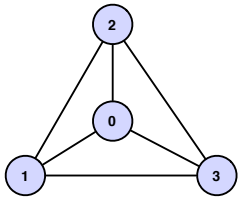

Figure 4: BGP policy cycles causing forwarding loops

4. *BGP policy cycles*: Figure 4 shows the classic "bad gadget" [25, 14] involving cyclic preference dependencies. Each of ASes 1, 2, and 3 prefer to route via its clockwise neighbor over the direct path to AS 0, and does not prefer a path of length 3. The routes will never stabilize because there is no configuration where no AS wants to change its route to AS 0. Furthermore, the system goes through many repeated states involving routing loops causing chronic unavailability.

Summary Some of the specific scenarios illustrated above can be alleviated using known solutions. For example, root cause notification (RCN) [33, 23] can prevent routing loops caused by link failures: in Figure 1, if 4 immediately floods the cause of the withdrawal—the failure of the link 4-5—to 2 and 3, they can prevent the loop. The example in Figure 2 can be addressed if 4 explicitly reveals its intended traffic engineering policy to 2, 3, and 6. However, it appears difficult to extend RCNs to prevent loops or blackholes induced by more sophisticated policy changes, as also noted by [33, 30]. As a result, most ISPs avoid interdomain traffic engineering whenever possible[1]. The iBGP example shown in Figure 3 can be alleviated using an approach similar to RCP that computes consistent routes in a logically centralized manner. Cyclic policy preferences as in Figure 4 are believed to be rare; Gao and Rexford [9] showed that if ASes restrict their policies to satisfy the "valley-free" and "prefer-customer" properties, such unstable configurations cannot occur. In practice, ASes may have more general policies that do not obey these restrictions [37].

Our goal is to design a single mechanism that can address all of these consistency problems in policy routing. Our solution, consensus routing, is a simple decentralized routing protocol that allows ASes to adopt general routing policies while ensuring high availability.

3 Consensus routing overview

The key insight in consensus routing is to cleanly separate safety and liveness concerns in policy routing. *Safety* means that a router forwards a packet strictly along the path adopted by the upstream routers unless if the adopted route encounters a failed link or router. Note that interdomain routing today does not satisfy this property, e.g., packets may traverse a loopy route even though no router has adopted that route. *Liveness* means that the network 1) reacts quickly to failures or policy changes, and 2) ensures high end-to-end availability defined as the probability that a packet is delivered successfully. By separating safety and liveness concerns, consensus routing achieves high availability while allowing ASes to exercise arbitrary routing policies.

Consensus routing achieves this separation using two simple ideas. First, we run a *distributed coordination* algorithm to ensure that a route is adopted only after all dependent routers have agreed upon a globally consistent view of global state. The adopted routes are *consistent, i.e., if a router A adopts a route to a destination via another router B, then B adopts the corresponding suffix as its route to the destination*. Note that consistency implies loop-freedom. Second, we forward packets using one of two logically distinct modes: 1) a *stable* mode that only uses consistent routes computed using the coordination algorithm, and 2) a *transient* mode that heuristically forwards packets when a stable route is not available.

In BGP, a router processes a received update using its policy engine, adopts the new route in its forwarding table, and forwards the change to its neighbors. In comparison, a consensus router simply logs the new route computed by the policy engine. Periodically, all routers engage in a distributed coordination algorithm that determines the most recent set of *complete* updates, i.e., updates that have been processed by every router whose forwarding behavior depends upon the updates. The coordination is based on classical distributed snapshot [5] and consensus [27] algorithms. The routers use the output of

the coordination to compute a set of *stable forwarding tables* (SFTs) that are guaranteed to be consistent.

Packet forwarding by default occurs in the stable mode using SFTs. When routing changes are caused by policy, the entire system simply transitions from the old set of SFTs to a new set of consistent SFTs without causing routing loops and packet loss. When the stable route is unavailable—because the next-hop router is not accessible due to a failure and the resulting update to recover from failure is incomplete—packet forwarding switches to the transient mode. The router explicitly marks the packet as a *transient packet* and fails over to heuristics such as backup routes, deflections, and detour routes to deliver the packet successfully with high probability.

4 Stable Mode

The distributed coordination proceeds in *epochs* and ensures that, in each epoch, all ASes have a consistent set of SFTs. The k^{th} epoch consists of the following steps, each of which is explained in detail in subsequent subsections:

1. *Update log:* Each router processes and logs route updates (without modifying its SFT) until some node(s) calls for the $(k+1)^{th}$ distributed snapshot.
2. *Distributed snapshot:* The ASes take a distributed snapshot of the system. The snapshot is a globally consistent view of all the updates in the system. Some of them may be complete, and some may still be in progress, e.g., updates that were in transit to a router when the snapshot was taken, or were waiting on local timers to expire before being sent to neighboring routers.
3. *Frontier computation:*
   (a) *Aggregation:* Each AS sends its snapshot report consisting of received updates, with some of them marked incomplete, to *consolidators*[1], a set of routers designated to aggregate snapshot reports into a global view. Tier-1 ASes are good candidates for consolidators.
   (b) *Consensus:* The consolidators execute Paxos to agree upon the global view, and use the view to compute the set of updates that are globally *incomplete*, i.e., not been processed by all ASes whose routing state would be invalidated by the updates.
   (c) *Flood:* The consolidators flood the set of incomplete updates I and the set of ASes S that successfully responded to the snapshot back to all ASes.
4. *SFT computation:* Each AS uses this set of global incomplete updates and its local log of received updates to compute its $(k+1)^{th}$ SFT, adopting routes carried in the most recent complete update (i.e., not in I) and only involving ASes in S.
5. *View change:*
   (a) *Versioning:* At epoch boundaries, each router maintains both the k^{th} and the $(k+1)^{th}$ SFT. Each packet is marked with a bit indicating which SFT must be used to forward the packet.
   (b) *Garbage collection:* ASes discard the k^{th} SFT after the $(k+1)^{th}$ epoch has ended, i.e., when the $(k+2)^{th}$ snapshot is called for and distributed.

Sections 4.1–4.5 elaborate the above five steps and 4.6 lists safety and liveness guarantees. ASes are assumed failure-prone but not malicious. We discuss the implications of malicious ASes in Section 7. Before that, we briefly comment on the feasibility of the approach.

First, not all ASes need participate in the protocol in order to ensure loop-freeness. A stub AS can not be involved in a loop since no AS transits traffic through it. So, only transit ASes ($\approx$ 3000) participate in the protocol, an order of magnitude reduction compared to the total number of ASes (≈ 25000).

Second, ASes do not send all received updates to the consolidators. Instead, they only send identifiers for updates received in the previous epoch, and the consolidators send back a subset of the identifiers corresponding to updates deemed incomplete. Our evaluation (§ 6) shows that the additional overhead due to consensus routing is a small fraction of BGP's current overhead.

Third, a small number of consolidators (e.g., about ten tier-1 ASes) suffice. The consolidators run a consensus algorithm [27] once every epoch whose communication overhead is a small fraction of the dissemination overhead above.

4.1 Router State, Triggers, Update Processing

4.1.1 Router State

A consensus router maintains the following state:

1. *Routing Information Base (RIB)*: stores for each prefix the most recent (i) route update received from each neighbor, (ii) locally selected best route, (iii) route advertised to each neighbor; this is identical to BGP's RIB.
2. *History*: stores for each prefix a chronological list of received and selected routes in the RIB. A *received update* is added to the *History* when an update is processed, and a *selected update* is added when the best path to a prefix changes.
3. *Stable Forwarding Table (SFT)*: stores for each prefix the next-hop interfaces corresponding to the stable routes selected for the current and previous epochs.

4.1.2 Triggers

Consensus routers maintain the following invariant: if a router A adopts a new route to a destination, then every

[1]Consolidators are equivalent to 'acceptors' in Lamport's 'Paxos Made Simple' parlance[28].

router that had received the old route through A has processed the update informing it of the change. Consensus routing uses triggers to maintain this invariant.

A *trigger* is a globally unique identifier for a set of causally related events propagating through the network. A trigger is a two-tuple: *(AS number, trigger number)*. The first field identifies the AS that generated the trigger. The second field is a sequence number[2] that is incremented for each new trigger generated by that AS.

In BGP, each update announces a route and implicitly withdraws the previously announced route. In consensus routing, each update additionally carries a trigger, and this trigger is associated with the route being implicitly withdrawn and replaced by the route announced in the update. The trigger essentially tracks when the implicit withdrawal is complete, i.e., whether all routers that had previously heard of the old route have processed the update. For uniformity, we assume that a BGP withdrawal message is essentially an update message announcing a *null* route (which withdraws the previously announced route without announcing a new one).

To maintain our invariant, we have the following rules for associating triggers with updates. An AS A generates a new trigger to be sent along with an update under these circumstances: 1) There is a failure of the adjacent link or the next-hop router in A's current route to the destination, 2) there is an operator induced local policy change that causes A to prefer another route to the destination than the one it is currently using, or 3) A receives a path update which it prefers even though its current path is still available. Otherwise, when the received update implicitly withdraws A's current path to the destination, A simply propagates the trigger associated with the received update (as shown in procedure PROCESS_UPDATE).

An AS marks a trigger as incomplete if it has not yet fully processed an update carrying that trigger. "Processing" an update means both running it through its local policy engine and reliably propagating the resulting update to its neighbors. Thus, any update that is waiting for the MRAI timer at an AS is not fully processed - meaning that the trigger associated with it is marked incomplete.

Incomplete triggers at the time of the global snapshot are excluded from the SFT for the next epoch. To ensure consistency of routes, an AS does not adopt a new route until it knows that the trigger associated with the corresponding update is complete.

4.1.3 Update Processing

The procedure PROCESS_UPDATE presents the pseudocode for processing and propagating updates and their triggers during each epoch. For simplicity, we assume that all updates are for a single prefix. (We discuss prefix aggregation in Section 4.8.2.) Upon receiving an update from AS B with trigger t, AS A does as follows:

PROCESS_UPDATE(B, r, t):
1. Add the update's trigger t to the local set of incomplete triggers I_A.
2. Process the update as in BGP. Let *old* and *new* be the best route to the prefix before and after the update. We define *next_hop* for a route to be the first AS in the route.
3. Add the *received update* (t, r) to the head of the *History* list. Consider the following cases:
   (a) *old.next_hop* is not B, and *new.next_hop* is not B: do nothing since the best route has not changed.
   (b) *old.next_hop* is not B, and *new.next_hop* is B: propagate *new* to neighbors with trigger t', where t' is a newly generated trigger. Add the *selected update* (t', new) to the start of *History*.
   (c) *old.next_hop* is B: propagate *new* to neighbors with unchanged trigger t. Add the *selected update* (t, new) to the start of *History*.
4. Remove t from I_A.

4.2 Distributed Snapshot

Routers transition from one epoch to another by taking a distributed snapshot of global routing state. The local image corresponding to AS A consists of the sequence of triggers H_A stored in A's *History*, and the set of incomplete updates I'_A. An update can be incomplete at an AS when the snapshot is taken because: (i) the update is being processed by the AS (and is therefore in I_A), (ii) the AS might have processed a received update, but the resulting update to a neighboring AS is waiting for the MRAI timer to expire, or (iii) the update is in transit from a neighboring AS.

To initiate a distributed snapshot, an AS saves its local state and sends out a special marker message to each of its neighbors. When an AS A receives a marker message for the first time, it executes the following procedure:

SNAPSHOT:
1. Save the sequence of triggers in *History* as H_A.
2. Start logging any triggers received on channels other than the one on which the marker was received.
3. Initialize the set of incomplete triggers I'_A to ϵ.
4. Add the set of triggers in I_A to I'_A; these triggers correspond to the updates currently being processed.
5. Scan the outgoing queues for updates waiting on MRAI timers to expire, and add their triggers to I'_A.
6. Send a marker to all neighbors.
7. Stop logging triggers on a channel upon receiving a marker on that channel.
8. Once the marker has been received on all channels, add logged triggers to I'_A. These correspond to updates in transit during the snapshot.

The above algorithm is essentially the Chandy-Lamport snapshot algorithm [5] and can be initiated by any AS

[2]Despite the similarity, the trigger is not a Lamport clock as trigger numbers do not strictly increase as they propagate through the network.

in the system. A consistent view is obtained even when multiple ASes initiate the snapshot operation concurrently, and we thus require each of the consolidators to initiate the snapshot based on locally maintained timers. The distributed state (H_A and I'_A) across all ASes is aggregated in order to compute a *frontier*, i.e., the most recent complete update at each AS, as described next.

4.3 Frontier computation

The frontier computation consists of three steps:

Aggregation: After the snapshot, each AS A sends to all consolidators the following *snapshot report*:

1. The set of incomplete triggers I'_A.
2. The saved sequence of triggers H_A.

Consensus: Typically, the Tier-1 ASes act as replicated consolidators. Replicated consolidators ensure that (i) there is no single point of failure, (ii) no single AS is trusted with the task of consolidating the snapshot, (iii) a consolidator is reachable from every AS with high probability.

The consolidators wait for snapshot reports for a certain period of time. They then exchange received snapshot reports so that reports that were reliably sent only to a subset of the consolidators are propagated to all consolidators. The message exchange does not guarantee that all consolidators have the same data. Paxos is therefore employed in order to reach a fault tolerant agreement on the set of ASes, S, that have provided I'_A and H_A. Consolidators propose a value for S by communicating the reports that they have received to a majority of consolidators. (This communication could be optimized to avoid the transmission of data that is already available at the recipients.) The majority then picks one of the proposed set of reports as the consensus value, and this value is propagated to the rest. Note that Paxos is safe, but not live: in no case will the consolidators disagree on the set of ASes S, but if the consolidators fail repeatedly (unlikely if they are Tier-1s), then progress may be delayed.

When the consensus protocol terminates, each consolidator uses the snapshot reports I'_A and H_A of each AS $A \in S$ to compute the set of incomplete triggers I in the network. This set I is computed using the procedure COMPUTE_INCOMPLETE which works as follows. A trigger is incomplete if present in an any I'_A. A trigger is said to *depend* on all triggers that precede it in *any* H_A. This property ensures that any causal dependencies between updates is captured by our system. A trigger t is defined *complete* only if neither t nor any trigger it depends on is incomplete. Therefore, if a trigger is incomplete, then all triggers that follow it in any H_A would also be considered incomplete.

COMPUTE_INCOMPLETE$(S, I'_A[\,], H_A[\,])$:
1. Initialize $I = \bigcup_{A \in S} I'_A$.
2. Do until I reaches a fixed point:
   (a) For each $t \in I$, for each A do:
      i. if t occurs in H_A, add the first occurrence of t and all subsequent triggers in H_A to I.

Flood: The set I of incomplete triggers in the network enables ASes to determine the most recent complete frontier. The consolidators flood the set of incomplete triggers I and the membership set S as computed above to all the ASes. At the end of this flooding phase, every AS uses the same global information about incomplete triggers I and the set of ASes S to compute the stable forwarding table for the next epoch. Note that a simple optimization here allows us to reduce the size of the flood message, by not sending the complete set I. Since the consolidators know the sequence H_A for each AS $A \in S$, they need to only send the first trigger from each H_A that is incomplete.

4.4 Building Stable Forwarding Table

After an AS receives the set of incomplete triggers I from the consolidators, it builds a new SFT and readies its state for the next epoch. The procedure for an AS A's router to build its SFT is as follows:

BUILD_SFT(I, S):
1. Copy the current SFT to be its previous SFT.
2. For each destination prefix p:
   (a) Find the latest *selected update* $u = (t, r)$ in p's *History* such that t is complete, i.e., t and all preceding triggers $\notin I$.
   (b) Adopt r as the route to p in the new SFT.
   (c) Drop all records before u from p's *History*.

Step 2 above adopts the most recent route update for a prefix such that the trigger associated with it is complete. If any adopted path contains an AS whose snapshot report was excluded by consensus, then the corresponding route is replaced by *null* in the SFT. This ensures that a slow or failed AS is not used to transit traffic in stable mode. Section 5 presents transient mode techniques that improve packet delivery in this case.

4.5 View change

Versioning: The end of BUILD_SFT marks the end of the k^{th} epoch and the beginning of the $(k+1)^{th}$ epoch. Since ASes do not have synchronized clocks, different ASes make this transition at slightly different times. To ensure consistent SFTs, ASes maintain and use both the k^{th} and $(k+1)^{th}$ SFT in epoch $k+1$.

Packet forwarding at epoch boundaries proceeds as follows. Once a router has computed the $(k+1)^{th}$ SFT, it starts forwarding data packets using the new routes. Along the way, if a packet reaches a router that has not finished computing the $(k+1)^{th}$ SFT, the router sets a bit in the packet header, and routers forward the packet

along the route in the k^{th} SFTs from that point onwards. (A single bit in the header suffices if packet transit times are less than the epoch duration. If not, we could use two bits in the header, and packets older than one epoch are forwarded using transient mode.) Once routers start forwarding a packet on the older route, the packet continues on that route until it reaches the destination. Disallowing the packet to switch back to routes in the $(k+1)^{th}$ SFTs ensures loop-free forwarding.

Garbage collection: ASes discard the k^{th} SFT after the $(k+1)^{th}$ epoch has ended, i.e., when the $(k+2)^{th}$ SFT has been computed. Discarding older tables ensures that slow or failed ASes do not consume excessive resources at other ASes. In the $(k+2)^{th}$ epoch, if an AS receives a packet sent using a route from the k^{th} epoch or before, it simply treats the packet as if the corresponding route were *null*, and switches to the transient forwarding mode.

4.6 Proof of Safety

Consensus routing generates consistent SFTs, i.e., *if a router A adopts a route $A...BP$ to a prefix through another router B, then B adopts the route P to that prefix*. Equivalently, two routes destined to the same prefix are defined consistent if they share a common suffix starting from the first common router. Consistency implies loop freedom.

The crux of the proof is that if an upstream AS picks a certain path in its SFT, then its immediate downstream AS must pick the corresponding suffix. If the downstream AS adopts another more recent path, then our system ensures that the withdrawal of the previous path is complete, ensuring that no upstream AS continues using the old path.

Theorem 1. *Consider the routes computed by two adjacent ASes (X and Y) to a given destination. If X applies a route Y.P (a path with next-hop Y) to its SFT, then Y applies the route P to its SFT.*

The proof of the above theorem is deferred to Appendix A

4.7 Liveness

Consensus routing satisfies the following property:

Theorem 2. *If an AS selects a sequence of routes $R_1, R_2, \ldots, R_i \ldots$ to some prefix, then for each i, it will eventually adopt some route $R_j, j \geq i$, in an epoch such that— i) none of the routers in R_j, and ii) no more than half the consolidators—have failed.*

The proof of the above theorem is deferred to Appendix B.

Note that this property implies that, irrespective of routing policies, ASes transition from one set of consistent policy-selected routes to another, under restricted failure assumptions.

4.8 Extensions

4.8.1 Multiple routers in an AS

Consensus routing, as presented above, can safely accommodate multiple border routers in an AS. Each border router plays the role that an AS plays above.Consistency safety is preserved as routers only adopt complete updates. However, this naive approach 1) does not scale well as some ASes may have several tens of border routers, 2) does not reflect the administrative unity of policies adopted by these routers.

To address these problems, consensus routing designates one (or more) router(s) in each AS as a local consolidator. The local consolidator collects the snapshot reports from each border router, generates a summary, sends the summary to the Tier-1 consolidators and receives back the set of incomplete triggers that it distributes to the border routers.

4.8.2 Prefix Aggregation

In order to reduce the size of routing tables in the Internet, BGP allows ASes to aggregate prefixes. Prefix aggregation in BGP works as follows: if an AS A is announcing paths to two adjacent prefixes, p_1/n and p_2/n, and p_1, p_2 differ only in their $(n-1)^{th}$ bit, then A can merge the two prefixes and announce only the larger prefix $p/(n-1)$, where p, p_1, p_2 are identical in the first $(n-1)$ bits. For example, prefixes "128.208.4.0/24" and "128.208.5.0/24" can be aggregated to "128.208.4.0/23".

Consensus routers maintains *History* on a per-prefix basis, and so the dependencies between different prefixes is not captured. Prefixes are usually independent except when aggregation occurs. Consensus routing can easily be extended to handle prefix aggregation. If an AS aggregates two prefixes p_1 and p_2 into a larger prefix p, then all subsequent updates received for p_1 and p_2 are also added to p's *History*. This ensures that an update to p completes only after the corresponding updates to its component prefixes (p_1 and p_2) have completed.

4.8.3 Transactional BGP

Consistency enables ASes to execute a sequence of updates atomically, e.g., an AS may wish to withdraw a prefix from one provider and announce it to another provider without intermittent disconnection or other unintended behavior. Atomic updates can also enable ASes to smoothly revert from BGP wedgies[12].

4.9 Protocol Robustness

We now describe how we deal with some of the problems that might arise in the face of failures:

AS fails to send its snapshot in time: For the epochs to progress smoothly, ASes have to send their local state to the consolidators in a timely fashion. The consolidators accept snapshotted states from other ASes for a period of time after they initiate/receive the snapshot message. When that period ends, the consolidators proceed to send each other the set of snapshots received so that all of them operate on the same information. If an AS fails to get its snapshot to any one of the consolidators, because either the AS is slow/misbehaving or all of the messages to consolidators are lost, then that constitutes a severe AS failure. In such cases, the unresponsive AS will not be used for forwarding traffic in the next epoch, especially since its state could be inconsistent with the rest of the network. The exclusion of such ASes is done by having the consolidators also send out the set of ASes S from whom snapshots were received, along with the consolidated list of incomplete triggers. ASes compute their SFTs as described earlier, but if any selected path contains an AS whose snapshot was not available, then that path is replaced by $null$ in the SFT. This ensures that the slow/failed AS is not used for transiting traffic in stable mode. In the next section, we present transient mode techniques that improve packet delivery in the face of such failures.

Consolidator fails: It is possible that the routers collecting the local snapshots from ASes may fail. This could mean that all the consolidators don't operate on the same information, especially if some snapshots are received only by the failed consolidator and they have been partially propagated to other consolidators. Our use of the Paxos consensus algorithm helps us ensure that all the consolidators operate on the same information with respect to ASes' local snapshots.

Recovery: An AS that does not reply to a snapshot request within a timely manner will be marked as failed. (Note that an AS is not allowed to reply to a snapshot if it has not completed the SFT computation for the previous epoch). It can be re-integrated similar to how BGP restores failed routers. The 'failed' AS exchanges paths with its neighbors, just as it would have had it recovered from a real failure in BGP. At the end of the epoch, if the corresponding triggers are complete, then the new SFTs computed can include the routers in the failed AS, at which point the AS is considered re-integrated. The introduction of new routers is identical to recovery of failed ASes.

5 Transient mode

Forwarding switches to the transient mode when a stable route is unavailable at a router. The stable route may be unavailable due to two reasons. First, the next-hop router along the stable route may be unreachable due to a failure. Routers will not arrive at a consistent response to the failure until the next snapshot. Second, the stable route may be null either because a non-null route has not yet propagated to the router, or some router was slow to submit a snapshot report.

Consensus routing enables a common architecture to incorporate several known transient forwarding schemes such as *deflection*, *detour*, and *backup* routes. Today, in both BGP and intradomain routing, transient schemes [39, 46, 23, 26] are believed necessary to maintain high availability in the light of fundamental limits on convergence times. What is lacking, especially in policy routing, is a mechanism that 1) reliably indicates when to switch to the transient mode and back, and 2) allows different transient forwarding schemes to co-exist. Consensus routing provides this mechanism ensuring that a packet strictly traverses adopted routes unless it encounters a failure, at which point it switches to a failover option in accordance with the AS's policy preferences.

5.1 Transient forwarding schemes

5.1.1 Routing Deflections

When a packet encounters a failed link, the corresponding router "deflects" the packet to a neighboring AS so that it traverses a different route to the destination. The router consults its RIB and identifies a neighboring AS that announced a different valid route to the destination. If the trigger corresponding to that route is complete, then the neighbor has applied the route to its SFT, and packet delivery is assured as long as the alternate route does not encounter other failed links. Multiple link failures can be handled by adding the identity of each failed link to the transient packet's header [26] before deflecting it.

If no neighboring AS has announced a different valid route to the destination, the router deflects the packet using *backtracking*, i.e., forwarding the packet back along the link on which it arrived. Backtracking may be extended to multiple hops if the previous AS does not have a different valid route in order to increase the likelihood of successful delivery. Caching information about failed links and pre-computing deflections corresponding to each next-hop further improves lookup times in the forwarding plane.

Unfortunately, backtracking alone is insufficient to guarantee reachability, even if physical routes to the destination exist. Figure 5 gives an example. Bold lines represent chosen best routes to the destination *D*. Each node exports only its best route to its neighbors. For example, node *1* knows two routes to *D*: *1-4-D* and *1-5-D*, but it exports to *S* only its best route *1-5-D*. *S* knows three routes to *D*: *S-1-5-D*, *S-2-5-D*, and *S-3-5-D*. Observe that all three routes go through the same link *5-D*. Thus, if the link *5-D* is down, then even backtracking all the way to

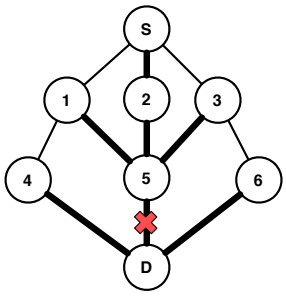

Figure 5: Why deflection packets need more information than BGP. Simple backtracking to provider ASes is not optimal.

S will not help packet delivery, as none of the nodes *S, 2, 5* know an alternate route. The next two transient forwarding schemes alleviate this problem.

5.1.2 Detour routing

When a packet encounters a failed link, the corresponding router selects an AS B and tunnels transient packets to it. Upon receiving the packet, B becomes responsible for forwarding the packet to the destination. If B is a Tier-1 AS, there is a high chance of delivering the packet to the destination, since Tier-1 ASes are likely to know of diverse routes to destinations.

This detour approach—having ASes off the standard forwarding path take responsibility for delivering a packet that encounters failures—suggests a potential new business model for ASes where certain ASes advertise themselves as available to deliver packets that encounter failures. Such ASes could either provide the tunneled detouring service for the entire Internet or just for designated prefixes. The tunneling service is similar in spirit to MIRO [45]. Exploring the business model for detour service providers in more detail is beyond the scope of this paper; our focus here is on evaluating the effectiveness of detour routing for transient packet delivery.

5.1.3 Backup Routes

When a packet encounters a failed link, the corresponding router uses a pre-computed backup route to forward the packet. One scheme for pre-computing backup routes to each destination is RBGP [23], which allows ASes to announce backup routes to each other with only slight modifications to BGP. RBGP also shows that choosing the backup route that is most link-disjoint from the primary route protects against *single link failures*—packet delivery is guaranteed as long as a valid route to the destination exists, under certain restrictions on the policies used for selecting and advertising routes. For instance, in the example discussed in Figure 5, node *1* would compute a backup route *1-4-D* and export it to *5*, thereby allowing *5* to use the route when its link to *D* fails.

5.2 Implementation Issues

Each of these transient forwarding methods require some changes to the way packet forwarding works today. Deflection routing requires additional space in the header to store information about the link failures encountered by the packet. Further, backtracking to the previous AS requires packet encapsulation, since the packet will be flowing against the forwarding tables of routers in the current AS. The router that encounters a failed link would have to encapsulate the packet and send it to the ingress router for the AS, which can then send the packet back to the AS it came from. In order to actually backtrack, one would have to keep track of which AS each packet came from, a task that would require additional processing even for stable mode forwarding. To avoid this overhead, backtracking can be approximated by routing the packet towards the source.

Detour routing also requires packet encapsulation, in order to tunnel the packet to the AS that is responsible for delivering those packets that encounter failures. Backup routing techniques like RBGP do not require changes to the packet format, but they require routers to maintain multiple forwarding tables, one for the regular paths and rest for the backup paths.

6 Evaluation

In this section, we explore the effectiveness of consensus routing in maintaining connectivity among ASes, and also analyze the overhead incurred by it. We evaluate consensus routing using 1) extensive simulations on realistic Internet-scale topologies, and 2) an implemented XORP [19] prototype. For simulation experiments, we built a custom simulator to compare the behavior of BGP and consensus routing with respect to consistency and availability in the face of failures and policy changes. Our results suggest that consensus routing yields significant availability gains over BGP while ensuring that packets traverse adopted routes, and that consensus routing adds negligible processing overhead.

6.1 Simulation methodology

We compare standard BGP and consensus routing by simulating routing decisions, protocol control traffic, and link failure on a realistic topology. We use the November 2006 CAIDA AS-level graph [17], gathered from RouteViews BGP tables [18], which includes 23,390 ASes and 46,095 links. CAIDA annotates the link with the inferred business relationship of the linked ASes, namely customer-provider, provider-customer, or peer-peer.

Our simulator simulates route selection and the exchange of route updates between ASes, accounting for MRAI timers. We use standard "valley-free" export policies matching economic incentives, whereby routes through peers and providers are exported only to cus-

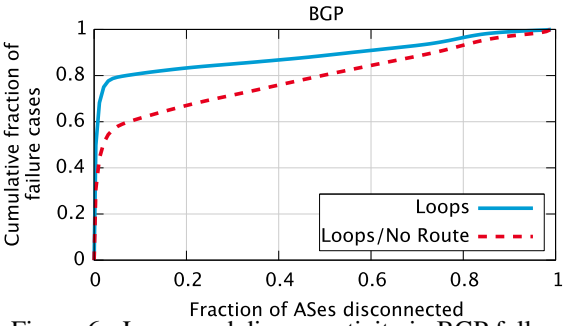

Figure 6: Loops and disconnectivity in BGP following a failure.

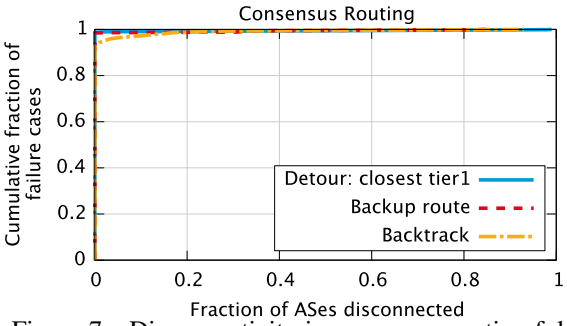

Figure 7: Disconnectivity in consensus routing following a failure.

tomers, and customer routes are exported to everyone [9]. We also follow standard route selection criteria, again driven by economic incentives, with customer routes preferred over peer routes, which in turn are preferred over provider routes. If multiple routes fall in the same category, the first tiebreaker is shortest AS path length, and the second is lowest next-hop AS identifier.

We study the routing protocols in three settings: (i) link failures, (ii) traffic engineering accomplished by announcing and withdrawing sub-prefixes, and (iii) traffic engineering accomplished by AS path prepending.

6.2 Link failures

To evaluate the ability of a protocol (BGP or consensus routing with one of the transient mode variants) to cope with failure, we set all routers in our simulator to use that protocol and allow them to reach a stable routing configuration. Then, we conduct a trial for each provider link L of each multi-homed stub AS A. A multi-homed stub AS is an AS with more than 1 provider and no customers; our topology includes 9,100 such ASes. We focus on these because the stub AS has a valid physical route to rest of the Internet even if the provider link L fails. In each trial, we fail the link L, and we record any AS that undergoes a period of disconnectivity to A before converging to a new route after the failure. We do not include ASes that the failure permanently disconnects and leaves without any policy-compliant routes. For BGP, an AS is disconnected if it has no route to A or if its forwarding path towards A includes a loop. For consensus routing, no loops occur, so an AS is disconnected if its SFT route to A includes the failed link, and the particular transient mode variant employed in the trial fails to find a valid route to A.

6.2.1 Standard BGP

We first simulate BGP and demonstrate that, following failures, disconnectivity is widespread before ASes eventually converge to new policy-compliant routes. Our simulator found convergence times similar to those of Internet measurement studies [25]. Figure 6 shows the prevalence of loops and other disconnectivity (note that a loop is a form of disconnectivity). It shows, for each failure, the fraction of ASes that at some point have no working route to the target AS, before eventually learning a new working route. We see that the failure of just a single link to a multi-homed AS causes widespread disconnectivity, with 13% of failures causing at least half of all ASes to experience routing loops, and 21% of failures causing more than half of all ASes to experience periods of disconnection.

The prevalence of loops argues for the safe application of updates, and the extensive disconnectivity argues for reliable routing in the face of failure.

6.2.2 Consensus routing with transient forwarding

We next evaluate the effectiveness of consensus routing, coupled with various schemes for transient forwarding, at addressing the BGP problems manifest in our simulations. By design, consensus routing ensures that routes adopted at epoch boundaries are loop-free. When adopted routes contain failed links, transient mode routing is invoked. We examine the effectiveness of the following representative schemes for transient forwarding in the face of failures.

Detouring through Tier-1: The router selects the closest Tier-1 AS and detours the packet there. If the Tier-1 does not have a route, it drops the packet.

Deflection by backtracking: The packet is backtracked to the source one hop at a time. If any hop has a failure-free route, it routes the packet to the destination. If the packet reaches the source and the source does not have an alternate route, it drops the packet.

Precomputed backup routes: Routers pre-compute backup routes as in RBGP, which picks the backup route that is most link-disjoint with respect to the primary route. When links fail, policy compliant backup routes are used.

Before invoking any of the above techniques, a router first consults its RIB to ascertain if any neighbor has announced a different valid route R in the most recent complete update. If so, the router safely deflects the transient packet to that neighbor known to have R in its SFT.

Figure 7 shows the effectiveness of consensus rout-

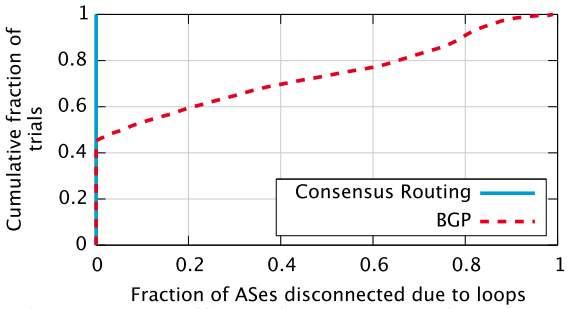

Figure 8: Traffic engineered subprefixes causes loops in BGP.

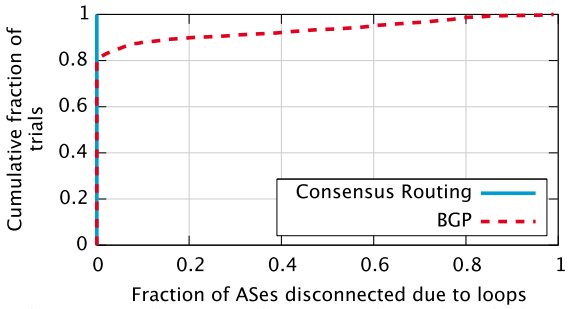

Figure 9: Path prepending causes intra-domain loops in BGP, leading to disconnectivity.

ing variants at finding usable routes in the face of failure. Even the simplest forms of transient routing greatly improve connectivity. Backtracking enables continous connectivity to at least 74% of ASes following 99% of the failure cases. More sophisticated failover techniques further improve connectivity. By detouring to a Tier-1 AS, all ASes maintain complete connectivity following 98.5% of the failures evaluated. Policy compliant backup routes provided similar benefits, with complete connectivity being maintained following 98% of the failures. This experiment suggests that consensus routing can accommodate existing transient forwarding schemes, along with one of the above transient mode variants would provide significantly higher levels of connectivity than BGP, for the failure cases we simulated.

6.3 Traffic engineering by using subprefixes

As described earlier (Figure 2), ASes can engineer traffic by advertising and withdrawing prefixes through a subset of its providers. If ASes advertise their entire range to all providers and selectively advertise their sub-prefixes to control routing, an AS can control its incoming traffic without losing the fault-tolerance benefits of multihoming. We now evaluate the impact of this form of traffic engineering on route consistency and availability. We evaluate standard BGP and consensus routing using our Internet topology graph, and consider subprefix-based traffic engineering for all multihomed ASes that have three or more providers. There are 3,451 such ASes and 12,890 inter-AS links between these ASes and their providers. In each run, we pick an AS and one of its providers, and withdraw the subprefix from each of the other providers. We then allow the system to converge using BGP and consensus routing, and examine the routes determined by the routing protocols during convergence.

Figure 8 shows the results. For BGP, we see that in more than 55% of the test cases, ASes were disconnected from the destination due to transient loops formed during convergence. Consensus routing transitions from one set of consistent loop-free routes to another, completely avoiding transient loops. Note that transient forwarding by itself does not help in this case, as there is no "failure" to trigger the use of transient forwarding or backup routes.

6.4 Traffic engineering by path prepending

We now examine the effect of AS path prepending on the consistency of the routing protocols. Path prepending is another form of traffic engineering wherein a multihomed AS controls the amount of traffic that flows in through each of its inter-AS links. By prepending itself multiple times on some of its advertised routes, an AS can make the prepended routes less preferrable, encouraging upstream ASes to reroute traffic to better paths.

For this experiment, we simulate the routing protocols using the same Internet topology as the previous experiments. In addition, we also model the interaction between BGP and iBGP as this form of traffic engineering is known to cause intra-domain routing loops that in turn lead to routing blackholes [43]. We therefore pick a Tier-1 AS (AT&T for our experiments) and expanded its corresponding node in the Internet graph to capture all of its internal topology, while continuing to represent all other ASes as single nodes. We obtained the internal topology for AT&T using the data collected by iPlane [29]. We assume that all the border routers are connected in a full-mesh topology, and model the propagation of routes through the AS according to the iBGP protocol. In this setting, an update is considered incomplete if it is not fully processed by all of AT&T's iBGP routers. For our evaluation, we consider all the multi-homed stub ASes present in the customer cone of AT&T. In each run of the experiment, we pick one of these stub ASes, which prepends itself three times[3]on routes to all but one of its providers. We repeat this run for each of its providers.

Figure 9 shows that in more than 20% of the cases, the Tier-1 AS suffers from intra-domain loops due to path prepending by some downstream AS and in many of those instances a significant fraction of other ASes lose connectivity to the target AS.

[3]Our analysis of BGP updates at the University of Washington showed that most origin ASes prepend their AS number thrice.

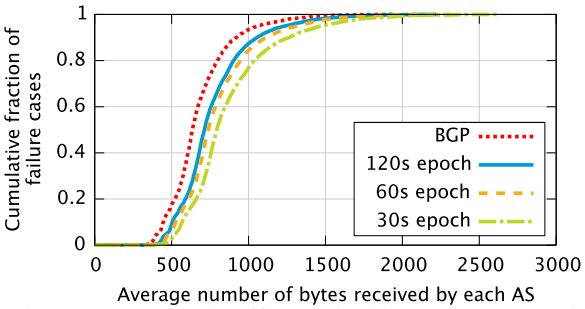

Figure 10: Control traffic required by consensus routing.

Number of nodes	Time when first node learns value	Time when last node learns value
9	434 ms	490 ms
18	485 ms	1355 ms
27	590 ms	1723 ms

Table 1: The time taken to run Paxos on 9, 18, and 27 PlanetLab nodes.

6.5 Overhead

The previous sections showed BGP's inability to maintain continuous connectivity in the face of failure or policy changes, as well as the significantly improved connectivity enabled by consensus routing. We now examine the additional cost incurred by consensus routing compared to BGP.

Volume of control traffic In addition to BGP route update messages, routers running consensus routing must send control traffic to take a distributed snapshot and flood the set of incomplete triggers. Figure 10 shows a CDF of the mean amount of control traffic sent by routers using BGP and consensus routing for our multi-homed stub AS access link failure simulations. The consensus routing overhead was computed for different epoch lengths: 30 seconds, 60 seconds, and 120 seconds.

We observe that the additional overhead introduced by consensus routing is rather negligible. This is because BGP sends updates that are relatively large to all the ASes, whereas the additional traffic sent by consensus routing is mostly sets of triggers, which are smaller; furthermore, this traffic is sent only to transit ASes.

Cost of consensus At the end of each epoch, the consolidators have to reach an agreement on the set of snapshots that will be considered for computing the SFTs. We evaluate the time taken for this process by running Paxos on PlanetLab. In our experiment, we pick 9 random PlanetLab nodes, (each representing a router in a Tier-1 AS), and run Paxos with each of these nodes playing the roles of a proposer, an acceptor, and a learner. We measure the time it takes from when a node proposes a value, to when it learns of the chosen value, and the results are available in Table 1. With 9 nodes, all them learn the agreed value in under 450 milliseconds. We then repeat the experiment with 18 and 27 nodes, which are settings that model replication by each consolidator AS to provide fault-tolerance. We observe that the slowest node learns of the chosen value in under 1.4 seconds and 1.8 seconds respectively. We believe that these overheads are acceptable for consensus routing even with epoch durations that are only a few tens of seconds. Further, the time for the consensus phase would be even lesser in a practical deployment, where the loss rates would be significantly lower than what we saw in our experimental PlanetLab setup. We would like to note that any other consensus algorithm, such as Virtual Synchrony [2], can be used instead of Paxos.

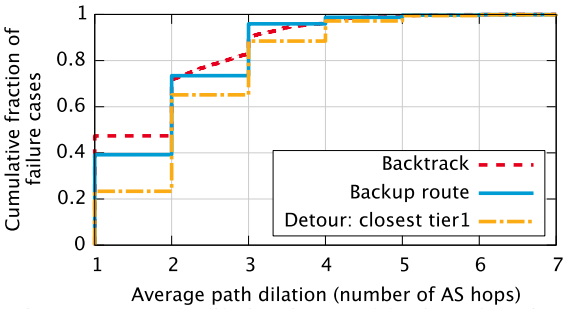

Figure 11: Path dilation incurred by interdomain transient mode options.

Path dilation Figure 11 shows the average amount of path dilation incurred by the various transient mode techniques. Path dilation is the length of the route (in AS hops) traversed by the packet when it encounters failure (from source to failure, then along the detour to the destination), minus the length of the pre-failure route from the source to the destination. It is a measure of how far packets have to be redirected. All the techniques experience a modest amount of dilation, with detouring to a Tier-1 experiencing the most (unsurprisingly).

Response time In BGP, an AS starts using a path as soon as it selects it. However, in consensus routing, an AS has to wait at least till the next epoch boundary before it can change the path it is using. We instrument our multi-homed access link failure simulations to identify the average delay between receiving a path and applying it to the forwarding plane in the case of consensus routing. Figure 12 shows the average delay for different epoch lengths. As expected, shorter epoch lengths allow quicker adoption of new paths. A 30 second epoch results in more than 90% of the paths being adopted in less than two minutes after the path is received. Note that typical BGP path convergence times are of similar duration.

Flux in adopted routes We also measure how many changes are made to routing tables across the entire topology for each routing failure event in our access link

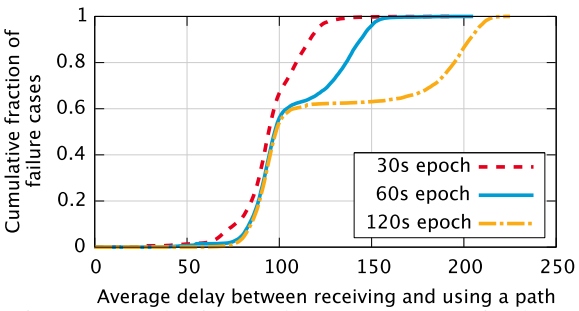

Figure 12: Delay incurred by consensus routing between receiving and using a path.

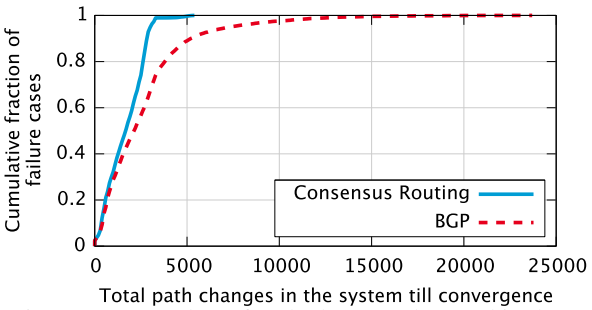

Figure 13: Number of path changes observed in the system.

failure simulations. In Figure 13, we see that ASes go through several route changes before converging on a stable route in traditional BGP, whereas the convergence takes place through fewer steps in consensus routing. For consensus routing, the number of changes per failure event is typically just one or two per AS, corresponding to the number of epochs the AS takes to stabilize to its eventual route.

6.6 XORP Implementation

Although simulation allows us to measure the additional message complexity or path dilation incurred by consensus routing, it is inadequate to measure more fine-grained processing overhead for each received update. To this end, we implemented a prototye of consensus routing on the eXtensible Open Router Platform (XORP)[19] software router. The actual task of implementing the protocol on a real router helped us refine the transition from design to implementation. For example, in order to ensure backward compatibility, it is essential that the format of the packet on the wire not be changed. Triggers must therefore be added to the BGP Update Packet either as a new optional transitive path attribute, or as a community attribute. While BGP routers ignore path attributes for withdrawals, consensus routing requires routers to examine the path attributes for all BGP Update messages. In addition, the implementation also helped identify the optimizations that reduced the overhead for flooding the set of incomplete triggers back to the transit ASes.

Implementation complexity Using lines of code as a rough approximation of implementation complexity, we find that our XORP implementation adds only 11% more lines to the default BGP implementation.

Computation requirements We use our XORP prototype to evaluate the computational requirements of consensus routing for a typical workload. Beyond the requirements of standard BGP, routers using consensus routing need to maintain *History* and track incomplete triggers. To measure the additional overhead, we replay a quarter-hourly log of routing updates collected by a RouteViews server to a machine running consensus routing. Because the logs do not contain triggers, we assign each update a new trigger. This part of the evaluation excludes the snapshot and flooding overhead. We conduct a trial for each of the quarter-hourly logs from a random day and find that the computational load of consensus routing for a trial is at most 8% more than for BGP.

7 Security Implications

The separation of packet delivery into two logically distinct modes of packet delivery — stable and transient – offers other potential benefits with respect to routing security, which we discuss next.

Much prior work has studied the problem of securing BGP against malicious ASes such as SBGP [21], soBGP [44], SPV [20], Listen and Whisper [42]. The primary focus of these efforts is to add cryptographic security to BGP, e.g., to ensure authenticity and integrity of advertised routes; these mechanisms are applicable to consensus routing as well. However, malicious ASes can also use non-cryptographic attacks that exploit protocol dynamics inherent to BGP. For example, a malicious AS can repeatedly announce and withdraw a route causing large shifts in the data plane at ASes much further away [31]. It can also abuse MRAI timers and route flap dampeners to intentionally slow down the responsiveness of other benign ASes to network conditions. MRAI timers are considered necessary to both reduce convergence time and message overhead, and opinions differ on whether or not there is any benefit to further tuning timers [13, 36, 7] even assuming a benign environment.

The stable mode makes routing more deterministic and hence easier to secure. ASes can impose an a priori agreed upon upper and lower bound on the interval between snapshots. While we have not implemented this yet, using standard byzantine fault tolerance agreement techniques [4], ASes that are slow or unresponsive either due to benign or malicious reasons can be consistently excluded from the snapshot observed by all ASes provided the network is sufficiently well-connected and the number of byzantine ASes is small. Thus, two good ASes will see the same set of complete updates at the end of an epoch ensuring that their SFTs are consistent. The

impact of spurious transient packets can be limited by the following mechanisms - (i) using the failure information only for that packet as opposed to caching it, (ii) limiting the rate at which transient packets can be generated by a single AS as well as in aggregate, (iii) detecting and punishing ASes generating conflicting transient packets. Finally, a malicious AS can still drop all the packets in the forwarding plane after announcing legitimate routes. End-to-end tecniques are needed to detect and punish such ASes. In fact, the consistency property—knowing the exact route a packet will take—makes this detection easier.

8 Related Work

Consensus routing, to our knowledge, is the first interdomain routing proposal that allows general routing policies while ensuring high availability. A large body of prior work contributed valuable insights to its design.

Route availability Several measurement studies highlight BGP problems such as slow convergence after routing changes accompanied by extended loops and packet loss [32, 25, 6, 16, 40, 43, 22]. Labovitz et al. [25] found that Internet routers may take several minutes to converge to a consistent view after a failure with a thirty-fold increase in packet loss during this period. They identified spurious updates during transient periods as a major cause for delayed convergence. Chang et al. [6] found that up to half of all updates are transient, i.e., adopted routes before and after an event are the same.

Studies at Sprint [16, 40] estimate that up to 90% of all packet loss is caused by interdomain routing loops. Furthermore, packets that escape loops experience experience substantial latency dilation ranging from 25-1300ms. Wang et al. [43] found that a single routing event can cause loops and multiple loss bursts lasting up to 20 seconds. Kushman et al. [22] analyzed VoIP quality and found that up to 50% of unintelligible voice samples correlated with BGP updates, with accompanying packet loss periods of 30 seconds or more.

Previous studies [25, 13] suggest that there are fundamental limits on how quickly a network converges after a routing event. The convergence time is determined by the diameter of the network times the MRAI timer [8] (believed necessary to prevent a superexponential message blowup and typically about 30 seconds long). Consensus routing recognizes these limitations and ensures that during convergence 1) packets do not loop , 2) transient forwarding maintains high availability, 3) consensus itself adds little additional overhead.

Complex dynamics Researchers and practitioners continue to discover complex and unintended effects of BGP dynamics. These include counterintuitive interaction of timers [30]; "wedgies" or unintended stable route configurations that are difficult to revert [12]; a recovering link causing routing blackholes due to the interaction of intradomain and interdomain routing [43], etc. BGP configuration is considered by some as a black art understood only by a precious few network gurus [41]. This state of affairs calls for making interdomain routing simple to comprehend and manage, which consensus routing achieves by enabling a consistent view of routing state.

Griffin et al. [14] showed that cyclic policy preferences can cause persistent instability. Gao and Rexford [9] showed that by restricting routing policies, stability is guaranteed without global coordination. Consensus routing uses distributed coordination to ensure that ASes transition from one set of consistent (loop-free) routes to another, even under general routing policies. The transient mode ensures high availability throughout.

Consistent Routing: Consistent routing solutions have been previously proposed for intradomain settings. A major advantage of an intradomain settings is that it allows for a logically centralized route computation and information dissemination plane as advocated by RCP [3] or 4D [11]. Luna-Aceves [10] proposed a decentralized approach to consistent route computation for shortest-path distance vector routing, where each node waits for an acknowledgment from the upstream node before adopting the route. The C-BGP proposal by Kushman et al. [24] is a natural extension of this approach to policy routing. Unfortunately, waiting for acknowledgments from upstream ASes means that ASes must rely on timeouts longer than the worst-case BGP convergence delay to ensure consistency; during this time the destination may be unavailable even though a policy-compliant physical route exists. Consensus routing also waits for the withdrawal of the old route before adopting the new route. However, the transient mode ensures high availability even when a stable route is unavailable. The stable mode ensures 1) consistency safety, and 2) progress when more than half the consolidators are up—the epoch length is determined by propagation delays and is not limited by BGP convergence delays. Further, consensus routing not only provides routing consistency, but it can also be extended to support atomic/transactional updates. Transactional updates enable operators to enact multiple traffic engineering operations simultaneously in order to smoothly transition from one set of stable policies to another without causing blackholes and other anomalies.

Triggers in consensus routing are similar to "root cause notification" (RCN) mechanisms proposed previously to weakly track causal relationships between propagating updates to speed convergence [33] or prevent loops [35, 23]. To our knowledge, consensus routing is the first proposal to adapt this mechanism for consistent routing under general policies.

Failure Recovery: The transient mode is similar in spirit

to similar proposals for intradomain routing [46, 39, 26]. The proposal by Lakshminarayanan et al. [26] for link-state routing maintains consistent routing tables and encodes information about failed links in packet headers to route around the failure.

R-BGP [23] proposes to use pre-computed backup paths to provide reliable delivery during periods where the network is adapting to failures. Our results on backtracking and finding detours on the fly indicate that packets can be delivered with a high probability even when backup paths are not known, given the current nature of the Internet graph. Furthermore, R-BGP relies on the provider>peer>customer preference for loop-freeness, whereas consensus routing computes consistent routes with general policies.

9 Conclusion

Networking researchers lay great store by the need to improve Internet availability from about two nines (99%) today to four to five nines like other infrastructural services. Although link failure rates can be reasonably expected to improve with time, and consequently drive down periods of transient unavailability to negligible values in intradomain settings, a policy-driven interdomain routing protocol is fundamentally susceptible to long periods of convergence causing transient unavailability. Improving Internet availability to five nines requires us to recognize this fundamental limitation and design around it. Our proposal, consensus routing, is a modest step towards that goal.

Consensus routing separates concerns of safety (or consistency) from liveness by using two logically separate modes of packet delivery. The stable mode leverages classical distributed algorithms to guarantee that routers always operate with a consistent view of the network, while the transient mode provides responsiveness and robust packet delivery in the face of failures. Our experimental results show that consensus routing can be implemented with a modest increase in implementation complexity and message overheads, while yielding a significant improvement in routing availability.

References

[1] Personal communication with Aspi Siganporia of Google Inc.

[2] K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. In *SOSP*, 1987.

[3] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and Implementation of a Routing Control Platform. In *NSDI*, 2005.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[5] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[6] D. Chang, R. Govindan, and J. Heidemann. The Temporal and Topological Characteristics of BGP Path Changes. In *ICNP*, 2003.

[7] S. Deshpande and B. Sikdar. On the impact of route processing and MRAI timers on BGP convergence times. *GLOBECOM*, 2004.

[8] S. Deshpande and B. Sikdar. On the impact of route processing and MRAI timers on BGP convergence times. In *Global Telecommunications Conference, 2004. GLOBECOM '04*, 2004.

[9] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM TON*, 9(6):681–692, 2001.

[10] J. J. Garcia-Luna-Aceves. A unified approach to loop-free routing using distance vectors or link states. In *SIGCOMM*, 1989.

[11] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5), 2005.

[12] T. Griffin and G. Huston. BGP wedgies. Network Working Group, RFC 4264, November 2005.

[13] T. Griffin and B. Premore. An Experimental Analysis of BGP Convergence Time. In *ICNP*, 2001.

[14] T. G. Griffin, B. F. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM TON*, 10(2), 2002.

[15] T. G. Griffin and G. Wilfong. On the correctness of iBGP configuration. In *SIGCOMM*, 2002.

[16] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Detection and analysis of routing loops in packet traces. In *IMW*, 2002.

[17] http://www.caida.org/data/active/as relationships/. The CAIDA AS relationships dataset, November 2006.

[18] http://www.routeviews.org. RouteViews.

[19] http://www.xorp.org. XORP: Open source IP router.

[20] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure path vector routing for securing BGP. *SIGCOMM CCR*, 34(4), 2004.

[21] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol. *IEEE JSAC*, 18(4):582–592, 2000.

[22] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?! It must be BGP. *CCR*, 37(2):75–84, 2007.

[23] N. Kushman, S. Kandula, and D. Katabi. R-BGP: Staying Connected in a Connected World. In *NSDI*, 2007.

[24] N. Kushman, D. Katabi, and J. Wroclawski. A Consistency Management Layer for Inter-Domain Routing. Technical Report MIT-CSAIL-TR-2006-006, MIT, 2006.

[25] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *SIGCOMM*, 2000.

[26] K. K. Lakshminarayanan, M. C. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.

[27] L. Lamport. The part-time parliament. *ACM TOCS*, 16(2):133–169, 1998.

[28] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4), 2001.

[29] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. An-

derson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.

[30] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz. Route flap damping exacerbates internet routing convergence. In *SIGCOMM*, 2002.

[31] O. Nordstrm and C. Dovrolis. Beware of BGP attacks. *SIGCOMM CCR*, 34(2), 2004.

[32] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM TON*, 5(5):601–615, 1997.

[33] D. Pei, M. Azuma, D. Massey, and L. Zhang. BGP-RCN: Improving BGP convergence through root cause notification. *Computer Networks*, 48(2):175–194, 205.

[34] D. Pei, X. Zhao, D. Massey, and L. Zhang. A study of BGP path vector route looping behavior. In *ICDCS*, 2004.

[35] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang. Improving BGP convergence through consistency assertions. In *IEEE INFOCOM*, 2001.

[36] J. Qiu, R. Hao, and X. Li. The optimal rate-limiting timer of BGP for routing convergence. *IEICE Transactions on Communications*, E88-B(4):1338–1346, September 2004.

[37] S. Y. Qiu, P. D. McDaniel, and F. Monrose. Toward valley-free inter-domain routing. In *IEEE ICC*, 2007.

[38] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen, and O. Bonaventure. Interdomain traffic engineering with BGP. *IEEE Communications Magazine*, 2003.

[39] M. Shand and S. Bryant. IP Fast Reroute Framework. IETF Draft, 2007.

[40] A. Sridharan, S. B. Moon, and C. Diot. On the correlation between route dynamics and routing loops. In *IMC*, 2003.

[41] J. W. Stewart. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, 1998.

[42] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and Whisper: Security Mechanisms for BGP. In *NSDI*, 2004.

[43] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. *SIGCOMM CCR*, 36(4), 2006.

[44] R. White. Securing BGP through Secure Origin BGP. *Internet Protocol Journal*, 6(3), 2003.

[45] W. Xu and J. Rexford. MIRO: Multi-path interdomain routing. In *SIGCOMM*, 2006.

[46] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, 2006.

APPENDIX

A Proof of Safety:

Theorem 1. *Consider the routes computed by two adjacent ASes (X and Y) to a given destination. If X applies a route Y.P (a path with next-hop Y) to its SFT, then Y applies the route P to its SFT.*

Proof. Assume that X applies $Y.P$ to its SFT. From procedure BUILD_SFT, we know that the last completed *selected update* in X's *History* (H_X) corresponds to the route $Y.P$. Let this record be $u(XYP) = (t_x, X.Y.P)$, where t_x is the trigger associated with the implicit withdrawal of the old path announced by X. Also Y must have previously advertised the path $Y.P$ to X. The corresponding *received update* $(t_y, Y.P)$ precedes $u(XYP)$ in H_X.

We now make the following observations:

1. Trigger t_y is complete, since $uXYP$ is complete and this trigger precedes $uXYP$ in H_X.
2. In H_Y, the *selected update* $u(YP) = (t_y, Y.P)$ is complete (since completeness is a global property). This follows directly from the previous observation.

We now show that Y chooses the route P for its SFT, using a proof by contradiction.

Suppose that Y applies to its SFT some route other than P Recall that procedure BUILD_SFT chooses a path based on the last completed *selected update*. Thus, there must be some *selected update* in H_Y that occurs after $u(YP)$. Let $Y.P'$ be the path that replaces $Y.P$, and let $u(YP') = ([t'_y, Y.P')$ be the first *selected update* after $u(YP)$.

Since we assumed that $u(YP')$ is complete, trigger t'_y must be complete. Further, since Y had announced the path $Y.P$ to X, when it changes its preferred path, it announces the change to X with the trigger t'_y. We know that in H_X, the *received update* with trigger t'_y occurs after the original announcement $u(YP)$. However, it could occur either before or after $u(XYP)$. We consider both cases, and show that both would lead to contradictions.

- *Case 1:* t'_y occurs before $u(XYP)$. If this happened, $Y.P$ would no longer be in X's RIB, so X's BGP engine would never announce the path $X.Y.P$, meaning that X wouldn't be allowed to choose the path $X.Y.P$, which contradicts our original asusmption.
- *Case 2:* t'_y occurs after $u(XYP)$ in H_X. Given that t'_y is complete, all records preceding it in H_X are also complete. This implies that there are no *selected update*s between $u(XYP)$ and t'_y, as $u(XYP)$ is the last completed *selected update*. Therefore, when t'_y is received by X, $X.Y.P$ is X's selected path, causing X to choose a new path. This change would result in the creation of a *selected update* $u(S)$ of the form (t'_y, S), where S is the new selected path. Note that the same trigger is retained since the path being implicitly withdrawn is the path being used currently. As t'_y is complete, the *selected update* $u(S)$ in H_X would be complete. This contradicts our original assumption that $u(XYP)$ is the last announcement record that is complete.

Therefore, it must be the case that $u(YP)$ is the last completed announcement record in H_Y, and so by BUILD_SFT, Y applies the route P to its SFT. □

B Liveness:

Theorem 2. *If an AS selects a sequence of routes $R_1, R_2, \ldots, R_i \ldots$ to some prefix, then for each i, it will eventually adopt some route $R_j, j \geq i$, in an epoch such that—i) none of the routers in R_j, and ii) no more than half the consolidators—have failed.*

Each of these route updates $R_1, R_2, \ldots$ is associated with a trigger. Let the corresponding sequence of triggers be $t_1, t_2, \ldots$. Consensus routing picks the last update with a complete trigger. We therefore need to show that for any i, t_i is eventually complete, thereby allowing R_i or some later route to be selected.

We begin by proving a few lemmas regarding trigger propagation, and then use them to prove the theorem.

Lemma 1. *If an existing trigger t does not occur in I'_A for any A in a particular epoch, then t does not occur in I'_A for any A in any subsequent epoch.*

Proof. $t \in I'_A$ implies that either A was processing the update carrying t or that the update was being transmitted to A, when the snapshot was taken. If no AS was processing an update with trigger t, then outgoing updates from any AS cannot have trigger t. This follows from step 3 of PROCESS_UPDATE where an existing trigger is retained only if an update is being sent out in response to a received update. □

Lemma 2. *An AS A can send out a trigger t at most once.*

Proof. A trigger is propagated unchanged by an AS if and only if the current route to the destination is being changed by the next hop in the route. In all other cases a new unique trigger is generated by the AS. Each trigger is associated with the old route being withdrawn. When an AS propagates a trigger, it changes its route to the destination, and the old trigger no longer applies to this new route. Changing to yet another route would result in a different trigger. Therefore, each AS can send out a trigger at most once. □

Lemma 3. *An AS A can declare a trigger $t \in I'_A$ only for a finite amount of time.*

Proof. A declares $t \in I'_A$ in an epoch only if the update containing t was in transit to A, or if it was waiting for the MRAI timer to expire before being sent out, when the snapshot was taken. If t was in transit when the state was saved, and had reached A when the snapshot finished, then it would not be in transit at the next epoch. The MRAI timer is finite, and so the update containing t would be sent out to the neighboring ASes within a finite time, and after that $t \notin I'_A$. □

Since we have a finite number of ASes, with each AS A being able to declare a trigger $t \in I'_A$ for a finite time, it follows that after a finite number of epochs, $t \notin I'_A$ for any A. Then, from Lemma 1, we know that $t \notin I'_A$ for any A at any subsequent epoch.

We have essentially proved that all ASes using a particular route receive the implicit withdrawal of the route within a finite time, i.e., t has finished propagating.

We are now ready to prove the theorem:

Proof. A trigger t can be considered incomplete for two reasons: (i) it hasn't finished propagating, or (ii) some trigger t' that occurs before t in some *History*is incomplete. Therefore, if all triggers that precede t have finished propagating, then t will be complete.

We have shown above that a trigger finishes propagation within a finite time, independent of other triggers. Now consider any trigger t. We know that t finishes propagation in a finite time. Also, every trigger that precedes t also finishes propagation in a finite time. Therefore, we have that t, and all triggers preceding t finish propagation in a finite time. t will then be complete.

Since we have shown this for any arbitrary t, it follows directly that for any i, t_i will eventually be complete, allowing an AS to pick R_i or some later route. This ensures that the system never waits on a particular trigger, and always makes progress. □