A Consistency Management Layer for Inter-Domain Routing

Nate Kushman
nkushman@mit.edu

Dina Katabi
dk@mit.edu

John Wroclawski
jtw@mit.edu

Abstract– This paper proposes an isolation layer – a shim – between inter-domain routing and packet forwarding. The job of this layer is to coordinate between Autonomous Systems (AS's) on when and how to modify the forwarding state to ensure inter-domain routing loops do not cause forwarding loops. The benefits of a consistency layer are twofold. First, it prevents the creation of transient inter-domain forwarding loops and the resulting packet loss, high latency, and connection failures. Second, by taking the burden of forwarding consistency off the inter-domain routing protocol, it enables inter-domain routing protocols with more complex convergence characteristics than BGP, such as protocols that optimize route selection based on performance. We offer two possible designs for the consistency layer. We prove that both designs are free of forwarding loops and show they are easy to deploy in the current Internet.

1 Introduction

Currently, inter-domain routing and forwarding are tightly coupled: any new route change is immediately pushed to the forwarding plane. As a result, when the inter-domain routing protocol exhibits transient loops, the forwarding state also suffers from the same loops, leading to packet loss, increased delay, and connection failures. In contrast, the circuit-based design of ATM and the centralized-routing model of RCP [3, 4] decouple routing from forwarding. While this paper also advocates this decoupling, it substantially differs from prior work in that it targets Internet inter-domain routing, where the routing is packet-switched and the route computation process is inherently distributed among the autonomous systems (AS's). It is particularly focused on mechanisms for ensuring the consistency of external (inter-AS) routes.

We propose an architectural modification to inter-domain routing. Our design inserts an isolation layer – a shim – between the routing plane and the forwarding plane. The consistency layer accepts routes from the control plane and pushes them to the data plane in a way that protects the consistency of the forwarding state across AS's. The consistency layer runs on the border routers, which communicate with border routers in other AS's to coordinate on when and how a new external (inter-AS) route should be pushed to the forwarding plane.

The benefits of the constancy layer approach are twofold:

(a) Addressing packet loss and delay caused by transient loops in today's Internet. Labovitz et al have shown that BGP, the Internet inter-domain routing protocol, suffers from transient loops and long convergence times, leading to significant packet loss and latency [9]. Hengartner et al have confirmed these results, showing that 90% of the packet losses in traces from the Spring network are caused by transient BGP loops [7].

Prior solutions to these problems have focused on reducing BGP convergence times [10, 2, 8]. While it is desirable to reduce BGP convergence times, there are inherent limits to how fast protocols can converge. These arise from the details of the protocol, the size of the network, and the dampening timers. A consistency layer directly addresses the data plane problems, i.e., packet loss and delay. It allows the routing protocol to converge uninhibited, but pushes only loop-free routes to the data plane during this convergence process.

(b) Enabling innovations in inter-domain routing and traffic engineering. The consistency layer takes the burden of forwarding consistency off the inter-domain routing protocol. As such, it allows network providers to explore more richer routing and traffic engineering options. We provide two examples:

1. It is desirable for inter-domain routing to react to performance metrics by moving away from routes with bad performance. However, such an adaptive protocol is likely to generate far more update messages, and as a result exhibit more transient inconsistencies. The consistency layer facilitates running such protocols in the Internet as it ensures that transient inconsistencies in the control plane will not cause forwarding loops.
2. Recent research has shown that a link state approach to inter-domain routing can substantially reduce the number of routing updates [13]. Yet, in a link-state-like protocol, each AS independently builds a map of the inter-domain topology and computes the best routes. In a network as large and diverse as the Internet, different AS's may take different times to collect the map and may have inconsistent views of the topology. Thus, if each AS pushes its routes to the data plane independently of other AS's, many forwarding loops may occur. A consistency layer can ease the deployment of such protocols and encourage the community to experiment with bolder inter-domain routing protocols.

The objective of this paper is to provide an initial discussion basis for the design of inter-domain forwarding consistency protocols. For the sake of concreteness, we focus on the problem of transient inter-domain loops and present our solutions within the context of BGP. However, the ideas underlying our consistency protocols are BGP-independent and work with other inter-domain routing protocols.

In the paper we first outline the concept and explore the design space of routing protocol consistency layers. We then offer two concrete approaches to the design of a consistency layer, and discuss the trade-offs inherent with each choice. Our first proposal, which we call C-BGP (Consistent-BGP), ensures that at any time each prefix has a consistent (i.e., loop-free) forwarding state. It works by enforcing an order on when the various AS's push a new route to the forwarding plane. In particular, it ensures that before an AS moves away from a path, all AS's forwarding traffic along that path have been informed of the route change, and have reacted appropriately.

Our second consistency protocol is called BGP-LP (BGP with Lingering Paths). A forwarding loop occurs when packets start their journey on a particular route, but get deflected downstream as they traverse an AS that moved away from the old route. Thus, forwarding loops can be avoided if each packet is delivered from sender to receiver entirely along the old route, or entirely along the new route. In contrast to C-BGP, BGP-LP allows an AS to immediately forward traffic on a new route even before other AS's have had the chance to move away from the old route. To do so, BGP-LP requires that after a route change, the AS continues to support the old route for a limited period. BGP-LP has a built-in mechanism to recognize when a packet started its journey on an old route. Such a packet is forwarded along the route it has started on.

We have evaluated C-BGP and BGP-LP using analytical methods. We have proven they ensure the forwarding plane is loop-free, while allowing BGP route updates to progress without deadlocks. The protocols are also fairly easy to implement and deploy in the current Internet. Our future work will focus on more clearly defining the architectural concept of the consistency layer, evaluating C-BGP and BGP-LP benefits using the current AS topology and logs of BGP updates, and exploring alternative designs and functional capabilities for consistency layer implementations.

2 Exploring the Design Space

The proposal of a consistency layer raises a few important questions. This section attempts to address some of them.

(a) Is transient inconsistency the exception or the norm for inter-domain routing? At its highest level our work is motivated by a single but sweeping change in perspective. Virtually all past routing work has assumed that convergence is the goal to be achieved, and that transient inconsistency is an unfortunate reality to reduce as much as possible. Our work begins the exploration of a different point of view. Fundamentally, we assume that in a sufficiently large network some part of the core routing computation is *always* in a transient state, and begin to explore ways to accommodate this fact – to treat transient inconsistency as the norm, not as an exceptional case.

(b) Should a consistency layer eliminate all transient loops? A consistency layer should eliminate transient loops for which the benefit of elimination is greater than the cost. We distinguish two cases. The first case occurs when an AS is switching between two paths that can both deliver packets. Here, avoiding the forwarding loop is quite desirable because the trade-off is enduring a sub-optimal route for a short period, vs. the elimination of transient loop packet-loss. We believe that this is the common case in the current Internet where BGP's path exploration often explores many available paths before settling on the final path, and where BGP is customarily used for traffic engineering [5, 11]. Additionally, as we consider the state of inter-domain routing in the future, it's clear that more reactive Traffic Engineering (TE) protocols and any negotiation based protocols will only cause additional updates between available routes. Lastly, it's worth noting that even if a particular set of BGP updates is caused by a link going down, most of the route changes can still be between paths that are physically able to route packets. In particular, a down link can cause an AS to move to using a different path, thus making available a new path that was not available before, and setting off a series of updates to move from one working path to a preferred one.

The second case occurs when an AS is switching away from a path that can no longer physically route packets. Here, the trade-off is less clear. The attempt at eliminating forwarding loops, may result in a period with no reachability. In this case we believe that the consistency layer should forego consistency to ensure that packets can be routed to an available path as quickly as possible. We note that past work has shown that through the use of root cause notification, this initial convergence period can be greatly reduced [10].

(c) Isn't it better just to reduce convergence time? Reducing convergence time can help reduce transient routing loops, and is quite desirable. But relying solely on solutions that reduce congestion time is problematic in two important ways. First, there is a limit to how far the convergence time can be reduced at the inter-domain level while still allowing AS's full control over their route selection, and without greatly increasing the communication overhead of the protocol. It has yet to be shown that transient loops at the inter-domain level can be reduced to a negligible rate while leaving AS's with an acceptable level of autonomy. Second, the requirement of fast convergence also greatly restricts the directions that inter-domain routing can be taken, significantly limiting innovation. In particular, it limits our ability to adapt inter-domain routing to performance metrics, as such adaptations will increase the update rate and the amount of transient inconsistency. Additionally, protocols such as those proposed from the game theory community to route traffic based on price [1, 6] may take much longer to converge than BGP, and cannot be considered for this reason.

(d) Are forwarding loops/instabilities an important problem? By injecting BGP updates into the network, Labovitz et al [9] have shown that even when no link has gone down, routing updates can cause up to 30% packet-loss for 2 minutes or more after a routing change. Also, two studies of packet traces from Sprint border routers have shown that up to 4% of the packets in such a trace are a part of a routing loop, and that greater than 90% of these routing loops are transient [12, 7]. Additionally, they found that in some traces up to 90% of the total packet-loss was due to congestion caused by routing loops, and that the load from these loops impacted the queuing delays of packets traversing the same router.

To confirm these results, and to directly compare the effect of routing updates on transient routing loops, we performed a small study of our own. Similar to Labovitz's study, we looked at packet loss rates near routing updates. However we also measured the rate of packet-loss specifically attributable to forwarding loops, by listening for ICMP `TTL Exceeded` messages. We measured these two packet-loss rates by sending packets every 30 seconds to the first (non ".0") IP address in every CIDR block (IP block) in the BGP routing table. To remove locations where the effect of BGP updates would be masked by congestion based packet-loss, we looked only at destinations with less than 10% packet-loss.

Fig. 1 plots the fractions of packet loss and TTL-Exceeded packet loss as a function of their time-distance from the closest BGP update for the same prefix. The figure shows high correlation between the occurrence of an update and a packet loss. In particular, near BGP updates (less than 10s from an update), one quarter of the packets are lost, and 10% of the packets are dropped particularly because they looped too many times. One should note that our results are inherently an underestimate of the impact of routing loops on packet loss because many routers do not send `TTL Exceeded` messages, and we have no way to measure the impact of the congestion created by the forwarding loop on non-TTL-Exceeded losses. Nonetheless, our results are consistent with past results, and confirm that forwarding loops have a significant negative impact on performance.

(e) Don't we also need to worry about intra-domain consistency? In this paper, we do not directly address intra-domain consistency. Current IGP protocols may exhibit transient loops but the scale and duration of such loops are much less than BGP loops [12]. Although a detailed study of intra-domain consistency is beyond the scope of this paper, our ideas, particularly those in §3.1, apply to any graph model of the network and thus should be applicable to IGP routing. Adding iBGP to the picture makes the problem more subtle and is left for future work. Approaches like RCP [3] may create the need for a consistency layer for intra-domain routing. Extensions of C-BGP to that environment should be plausible.

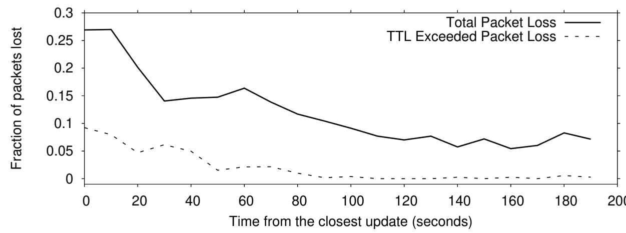

Figure 1—Fraction of lost packets and TTL-Exceeded losses as a function of the time between the occurrence of the loss and the closest BGP update

3 A Consistency Management Layer

The consistency layer is a distributed protocol that runs on the border routers. It can be thought of as taking data plane update requests from the routing protocol, and then performing these updates in a way that ensures packets are always forwarded along a loop-free path to their destination AS. As discussed in section §2, we make the design decision to forgo consistency guarantees when its current route can no longer deliver packets (is physically unavailable).

In order to keep the consistency layer as lightweight as possible and reduce its communication overhead, we piggyback consistency messages on the BGP updates. Though this may virtually blur the distinction between the routing protocol and the consistency layer, it allows us to concretely describe the consistency protocols, analyze them, and show that they are easy to deploy.

We present two possible designs for the consistency layer. These protocols target one type of routing inconsistency: data plane forwarding loops. It is future research to extend these to other forms of routing inconsistencies. In the rest of this section, we describe these algorithms in detail, show they prevent forwarding loops and do not deadlock, and discuss the resulting trade-offs.

3.1 Consistent-BGP

Our first approach is called Consistent-BGP or C-BGP.

3.1.1 Intuition

Transient routing loops can occur when an AS pushes a route change to the data plane, but other AS's, unaware yet of the move, try to send packets on the old route. C-BGP works by enforcing an order on when the various AS's push a new route to the forwarding plane. It ensures that before an AS moves away from a path, all AS's forwarding traffic along that path have been informed of the route change, and have reacted appropriately.

C-BGP has three types of messages: update messages, acknowledgement messages, and timeout messages. It continues to generate and send update messages exactly as BGP does. Syntactically these messages are exactly the same as BGP updates, except for the addition of a unique identifier to each message and the addition of a *forced* bit that we will explain below. However, because these messages act as announcements before an update has happened, their meaning differs slightly. Additionally, at the communication level, the only difference between BGP and C-BGP is that C-BGP sends acknowledgment messages to each update and timeout messages if acknowledgements take too long to arrive. Once an AS can ensure that all AS's using a path through it to get to the destination have both heard the update and appropriately reacted to it, it performs the update and sends a small acknowledgment message containing only the unique identifier of the update it is acknowledging. If acknowledgements do not arrive in a timely manner, then the AS performs the update anyway, and sends a timeout message indicating that the update has already been performed.

Thus, with C-BGP, the control plane convergence happens *exactly* as it does today, C-BGP does not slow down this convergence process at all. Additionally, the data plane will converge within the time it takes the control plane to converge, plus the time it takes to process and send acknowledgement messages on $O(d)$ machines, if d is the diameter of the network. This additional delay is similar to the time that it currently takes BGP withdrawal messages to reach the entire network.

3.1.2 *Algorithm Description*

We explain C-BGP at a high-level. The reader can find the protocol details and analysis in the appendix. For simplicity, the entire discussion is within the context of a single CIDR block (a single prefix). We distinguish between two cases.

Case 1: The current route is still available. When an AS decides to move away from the current path, it sends an update with the new path to the appropriate neighboring AS's (the neighbors to whom the policy allows exporting paths). The AS declares the routing update as *pending* and waits to receive acknowledgements from all of its neighbors. An AS that receives a routing update from a neighbor checks whether it is currently using that neighbor to send traffic to the updated prefix. If not, the recipient AS immediately acknowledges the update message, and updates its own routes according to BGP rules. However, if the AS is using the neighbor to reach the prefix then it needs to inform its own upstream AS's of the route change before it can acknowledge the update. First the AS decides how to change its own route in reaction to this update. Then, it generates its own update message, broadcasts it to its neighbors, and waits for the acknowledgements. Once an AS receives all the acknowledgements for a pending update, it can push the new route to the forwarding plane and acknowledge any downstream AS's.

The above description ignores an important complication: what happens if while waiting for an acknowledgement, the AS receives a new update for the same prefix? Here we emphasize the fact that BGP update messages will be sent in C-BGP, exactly as they are with BGP, i.e., according to the BGP route selection algorithm. However, the BGP decision algorithm runs as if all pending updates have already been committed. Thus, when an update is received, the normal BGP algorithm is run to see if an update should be performed in response, and if so, the update is put into a pending state, and the AS sends new update messages to its appropriate neighbors. This leaves only the question of when the acknowledgements are sent and the data plane is actually updated. In order to avoid deadlock and ensure that an update does not get starved, acknowledgements for pending updates are sent as soon as all of the up stream AS's have sent their acknowledgements. However, in order to ensure consistency, updates to the data plane are performed only once all acknowledgements have been received for the pending update, and all earlier pending updates.

Finally, we need to deal with the special case when the AS receives an update whose AS-path contains the AS receiving the announcement. These are the updates that could cause routing loops, and so they must be treated specially. To begin with, the standard case algorithm described above is performed on these updates. In addition, however, when such an update is received from a neighbor, any pending updates to move to that same neighbor are canceled (because it will cause a loop). A pending update is canceled by removing it from the pending update list, and associating all its pending acknowledgements with the following update (in the list of pending updates). No additional special messages are required, because the normal BGP decision making algorithm will make sure that an AS moves away from using a path that contains itself.

Case 2: The current route becomes unavailable. In the case the current route is physically unable to route packets (because of a down link, as opposed to a policy based withdrawal), the consistency layer should avoid slowing down changes to the data plane because, as discussed in section 2, the cost of consistency is too high. Instead, the consistency layer should use the normal uncoordinated method of data plane update, until all AS's previously using the down path have moved away from it. In particular, the AS with the failed path will immediately update it's data plane, without waiting for an acknowledgement and then send an update with the *forced* bit set. If an AS receives an update with the *forced* bit set for a path that it is currently using, then it immediately performs any necessary updates, and sets the *forced* bit on any updates it sends as a result. If however an AS receives a forced update for a path that it is not currently using, then it simply follows the Case 1 algorithm, and does not set the forced bit on any updates that it sends. It's important to recognize that the result of this algorithm is that Case 2 only affects the first round of updates for AS's who were using a path that has a link go down. Once the forced updates have reached all nodes, all other updates that result from this initial set of updates will follow Case 1.

3.1.3 *Misbehaving AS's*

The previous discussion assumed that all AS's will always correctly follow the protocol. Internet protocols however, must be designed with exactly the opposite in mind. There are two ways in particular that this assumption could be broken. The first is some kind of problem, and the second is malicious gaming. We'll attempt to deal with the first and help with the second. We consider the game-theoretic aspects of the consistency layer to be an interesting area for future research, however, they are out of scope for this paper, and we will not discuss them in any detail. It's clear however that in order for C-BGP to be practical it needs some way to deal with AS's that do not acknowledge updates.

We propose dealing with this problem through timeouts associated with each pending update. Upon timing out on a particular update, the AS would act as if all the acknowledgments had been received for that update. The AS would then send out another message to all the AS's to which it sent the original update containing only the message ID of the original message. Similar to forced updates, upon receiving such a message, an AS will act as if it timed out the update as well, perform any updates, and forward on the message to the appropriate neighbors. Finally, in the absence of malicious behavior, the timeout rule is unlikely to be invoked often as BGP updates are sent using TCP, which provides reliable communication.

3.1.4 *Consistency Analysis*

Our analysis shows that C-BGP achieves its goal in preventing transient loops, and that the execution of one update cannot be held up by any other updates occurring on the network. In particular, in the appendix, we prove the following:

THEOREM 3.1. *C-BGP prevents forwarding loops if Case 1 is followed for all updates.*

THEOREM 3.2. *In C-BGP, all updates continue to make progress, and an update cannot be slowed down by other updates happening contemporaneously.*

3.2 BGP-LP

Algorithms like C-BGP are fairly easy to implement in the current Internet architecture. But, they force an update to wait until it can ensure no transient loops. Below, we describe an alternative and more responsive approach to inter-domain forwarding consistency.

3.2.1 *Intuition*

The fundamental problem causing transient forwarding loops is that an AS pushes an update to the data plane, but other AS's are not yet aware of that change. We can imagine the AS's that have been announced a path through an AS, A, as arranged in a tree starting from A. If we imagine the branches of this tree follow the update messages, then when an update is pushed to the data plane, the announcement of that update is propagated along that tree. If the update has not yet reached the entire tree however, then there exists an *update horizon* along each branch of the tree, representing the boundary between the farthest AS that has heard the update, and the closest AS that has not yet heard the update. We call the period between update horizons an *update incarnation*, and observe that if a packet is forwarded at each router using the information derived from a single update incarnation it will follow a consistent path through the network. In the general case, there may be several updates active at the same time, and thus several update horizons moving through the network at the same time.

Routing loops can occur when the AS's on the opposite sides of an update horizon have different views of the network. When packets start their journey they are forwarded along a particular path, using the information from a specific update incarnation. But they may at some point cross an update horizon and be deflected to a second path, created by a later update incarnation. It is quite possible that the second path points back at an AS along the first path, causing a forwarding loop.

These loops could be avoided if the forwarding process at each router could distinguish between packets that originated within different update incarnations, and consistently forward each packet using information from that packet's original update incarnation–as long as that path is available. Intuitively, this implies that the data plane must maintain multiple active routes, one for each active update incarnation.[1] In contrast with the C-BGP protocol, this approach offers superior update performance by allowing multiple update incarnations to be active at the same time, at the cost of the extra state required to maintain the additional information. This state can be located in either the data packets or the forwarding tables.

3.2.2 *Implementation Details*

The simplest way to ensure that each packet is forwarded consistently is to collect the information from a single update incarnation at the packet origin and add it to the packet as a (AS-level) source route. This approach is well understood and would in theory be easy to implement because BGP naturally provides the appropriate path vector. However, it suffers from the packet size and performance overheads and perceived security drawbacks of IP

[1] An update incarnation becomes inactive when the information from later incarnations has propagated to all parts of the network.

source routing.

An alternative is to use the information propagated by each update horizon to build some form of label switched path (LSP) for that update incarnation through the network. In essence, AS's would need to maintain multiple source routes, one for each active update incarnation, through their networks and across boundary points. Packets entering the network within a particular update incarnation at their origin AS would remain on the source route associated with that update incarnation throughout their passage through the network. They would using the label information to identify at each AS boundary the next hop AS associated with the correct update incarnation. The cost of this implementation approach is the extra state associated with per-update-incarnation LSPs. Note that in practice the path labeling can be implemented with a variety of mechanisms such as MPLS or IP-level tunnels; our use of labeled path as a concept does not mandate a specific implementation.

4 Discussion & Future Work

This paper has presented a new architectural component for inter-domain routing. In particular, it has proposed the introduction of a new protocol layer between the routing layer and the data plane in order to ensure the consistency of the data plane. Additionally it has proposed two implementations of this consistency layer that resolve specificially the inconsistencies causing forwarding loops. There is much to be done, however, both on the architectural abstraction, and on the implementation itself.

This work has just begun to explore the details of what a consistency layer abstraction would actually look like. In particular, we need to explore the trade-off between efficiency and the crispness of the abstraction between the routing protocol and the consistency layer. This abstraction needs to be clarified in order to maintain efficiency while ensuring that it is logically independent of the routing protocol.

Additionally, a key future area of research with respect to consistency layer functionality is the possibility of dynamically trading off the benefits of consistency with the cost to achieve that consistency. So far, we have only considered distinguishing between the case where the current path is physically available, and the case where it is not. The set of possible trade-offs here is rich, including information about available routes, location in the Internet (edge vs. core), actual packet congestion and loss metrics, and a host of other factors which may affect this trade-off.

Lastly, we have considered only forwarding loops in this paper, but routing inconsistencies can also cause an AS to believe that it has no route to the destination even though a neighboring AS could easily route the packets. We believe that the consistency layer model can be extended with efficiently encoded back-up paths to deal with these types of inconsistencies as well.

References

[1] M. Afergan and J. Wroclawski. On the Benefits and Feasibility of Incentive Based Routing Infrastructure. In *Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, pages 197–204. ACM Press, 2004.

[2] A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved bgp convergence via ghost flushing, 2003.

[3] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and implementation of a routing control platform. May 2005.

[4] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers, 2004.

[5] N. Feamster, J. Borkenhagen, and J. Rexford. Controlling the impact of bgp policy changes on ip traffic, 2002.

[6] J. Feigenbaum, R. Sami, and S. Shenker. Mechanism design for policy routing, 2003.

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

[8] L. Jiazeng, X. Junqing, H. Ruibing, and L. Xin. An approach to accelerate convergence for path vector protocol. In *Globecom*, 2002.

[9] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *SIGCOMM*, pages 175–187, 2000.

[10] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. F. Wu, , and L. Zhang. Improving bgp convergence through consistency assertions. In *In Proc. IEEE INFOCOM. IEEE,*, 2002.

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

[12] A. Sridharany, S. B. Moon, and C. Diot. On the correlation between route dynamics and routing loops.

[13] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A Next-generation Interdomain Routing Protocol. In *ACM SIGCOMM*, Philadelphia, PA, August 2005.

APPENDIX

A A Consistency Shim

A.1 Consistent-BGP

Intuitively, an update in C-BGP can be performed to the consistency shim, or to the data plane itself. Updates that have been performed to the consistency shim, but not yet performed on the data plane are maintained in queue. Entries in this queue maintain:

ASPath This is the ASPath to be used when this update is commited to the data plane. This not neccessarily the ASPath of the original update, but the ASPath that will be moved to as a result of this update. Note that if the next hop AS is not changing then such an update may not require a change to the data plane.

Pending Acknowledgements This is the set of message update acknowledgements that will be sent once the all neccessary acknowledgements are recieved and the ASPath has been commited to the dataplane.

For ease of explanation, we will assume that each AS has only a single router[2].

There are two types of events that an AS can receive, Update events and acknowledgment events. These are outlined below with pseudo-code in the PROCESS-BGP-UPDATE and PROCESS-LAST-ACKNOWLEDGEMENT procedures respectively. The rest of the pseudo-code gives more detail on the procedures used by these two. For simplicity, the entire discussion is within the context of a single CIDR block. To clarify the pseudo-code, THIS-FONT will be used for names of pseudo-code procedures that are located elsewhere in the document, and *This-Font* for data structure members. Additionally we will consistently use a few variables names throughout the pseudo-code. These can be thought of as global variables:

A = the current AS
Q = the update Queue
U = the update just recieved
K = the acknowledgement just recieved

Lastly, we'll consistency use the data members *NextHop* and *ASPath* to refer respectively to the first AS in an ASPath, and the ASPath itself, associated with an update message, or with a queued update.

Upate Messages

When an update is recieved at the control plane, PROCESS-BGP-UPDATE is called:

1 PROCESS-BGP-UPDATE(*update U*)

```
 1  if no queued or current use of a path thru U.NextHop
 2      Acknowledge U
 3      if BGP would prefer U.ASPath over Q.Tail.ASPath
 4              QUEUE-UPDATE(U.ASPath, NULL)
 5  elseif U.ASPath contains A
 6      REMOVE-ALL-QUEUED-MOVES-TO(U)
 7  elseif Q.Tail.NextHop == U.NextHop
 8      P = BGP's new best ASPath given update U
 9      QUEUE-UPDATE(P, U.ack)
10  else
11      APPEND-TO-LAST-MOVE-AWAY(U)
12      if BGP would prefer U.ASPath over Q.Tail.ASPath
13              QUEUE-UPDATE(U.ASPath, NULL)
```

[2]It can be shown that the protocol generalizes to the case where there are many routers

In PROCESS-BGP-UPDATE, if *A* is not currently using a path through *U.NextHop* and has not queued any updates to move to a path through it, then *A* will acknowledge the update immeadiately. Then, using the normal BGP path preference ranking algorithm, it will consider whether it would like to move to the newly available path. If *A* decides that it would like to move to *U.ASPath*, then it sends out the corresponding BGP update, and queues the update to the data plane to be executed once all the acknowledgements are recieved (as outlined in QUEUE-UPDATE).

2 QUEUE-UPDATE(*ASPath P*, *ack K*)

```
1  Generate a new queue entry E with path P
2  Associate acknowledgement K with E
3  Add E to the queue
4  Generate new BGP update msg. V with ASPath P
5  Send V to all appropriate neighbors
```

If however *A* is currently using a path through *U.NextHop*, or has queued an update to move to a path through it, then there are three cases. The first case (lines 5 to 7) is the special case where *U.ASPath* contains *A*, and moving to this path would cause a routing loop. In this case, *A* will revoke all currently queued moves to use paths through *U.NextHop* and wait to acknowledge the update until it is no longer using a path through *U.NextHop*. This special case is outlined in REMOVE-ALL-QUEUED-MOVES-TO:

3 REMOVE-ALL-QUEUED-MOVES-TO(*update U*)

```
 1  for entries e in Q
 2      if U.NextHop == e.NextHop
 3              append all of e's pending acknowledgements to
                the previous entry in the queue
 4              remove e from the Queue
 5  if A is not currently using a path through U.NextHop
 6      Acknowledge U
 7  elseif A has no queued updates
 8      P = BGP's new best ASPath given update U
 9      QUEUE-UPDATE(P, U.ack)
10  else
11      APPEND-TO-LAST-MOVE-AWAY(U)
```

In the second case (lines 7 to 10), the ASPath in the update does not contain *A*, but the most recently queued update, *Q.Tail*, would update *A* to using a path through *U.NextHop*. Since the BGP algorithm is always running on the most recently queued update, in this case, *A* runs the normal BGP algorithm to determine whether it would like to continue using the path through *U.NextHop* despite the update, or if it would like to move to a different path. Either way, it queues an update to whatever is the new best path.

In the third case (lines 10 to 13), *A* has queued a move to a path through *U.NextHop*, but has later queued a move away from paths through *U.NextHop*.Since it is already announced it's move away from paths through *U.NextHop* to the appropriate neighbors, there is no need to send another announcement. However, A cannot acknowledge U until it has actually performed this move. Thus, the acknowledgement is associated with the last queued update which moves away from a path through *U.NextHop*. This association is outlined in APPEND-TO-LAST-MOVE-AWAY.

However, in order to both ensure that there are no routing loops, and that starvation does not occur, the algorithm needs to make a special distinction for updates to paths that are not yet being used. To better understand this, let's take an example where an AS, B, is currently using path $B{:}C{:}P_1$[3]. AS C then announces a transition from $C{:}P_1$ to $C{:}D{:}P_2$, and as a result of this B announces a transition to $B{:}E{:}P_4$. Before either of these updates occurs however, D announces a transition from $D{:}P_2$ to $D{:}P_3$. Let's assume that AS C would like to continue to move to using a path through D despite this update.

At this point C has two options. (1) Since it has not yet moved to using the path through D, it can immeadiately acknowlegde the update, announce the change upstream, and then not actually move to using D until it has recieved the acknowledgements for the update to $D{:}P_3$. (2) Alternatively, C can annouce the update from $C{:}D{:}P_2$ to $C{:}D{:}P_3$ to the AS's upstream of it, move to using a path through D as soon as it recieves acknowledgements to $C{:}D{:}P_2$, and then only acknowledge D's update when it recieves acknowledgements of $C{:}D{:}P_3$ from the AS's upstream of it. If (1) is used, then C may be starved from ever moving to using a path through D in the case where D is continuously updating it's path. Thus C must announce $C{:}D{:}P_3$ to B and wait for acknowledgement from B before it can acknowledge $D{:}P_3$ to D. Thus we must decide what B should do when it recieves such an announcement from C. Naively, B cannot acknowledge any update to C's path until it has moved away from using a path through C, and so it would wait to acknowledge $C{:}D{:}P_3$ until it had transitioned to $B{:}E{:}P_4$. However, since it will never use a path through D, it actually doesn't care if D updates it path from P_2 to P_3, and so this creates an artifical dependancy. And in fact, it can be shown that such an artificial dependancy can cause a deadlock in the algorithm. For reasons of brevity, we will not give an example of this here.

In order to avoid this problem, we take advantage of the fact that within a given AS, updates to the data plane are not performed until all previous updates have been acknowledged. Thus we recognize that B can immeadiately acknowledge $C{:}D{:}P_3$ because it knows that C cannot move to using a path through D until B has acknowledged $C{:}D{:}P_2$. In this case, we'll call the acknowledgement to $C{:}D{:}P_2$ an **enabling acknowledgement** for $C{:}D{:}P_3$ because C cannot move to $C{:}D{:}P_3$ until $C{:}D{:}P_2$ has been acknowledged, even if $C{:}D{:}P_3$ has already been acknowledged. More generally, acknowledgement K is said to be an *enabling acknowledgement* for update U, if path *UASPath* cannot be used until K is sent. We recognize that for an AS, A, enabling acknowledgements for a given update, U, include only acknowledgements for updates that were both sent before U, and originated at the same AS as U, or at an AS closer to A, than the as originating U. This special case is outlined in APPEND-TO-LAST-MOVE-AWAY on lines 7 through 9.

[3] We'll use the notation $B{:}C{:}P_1$ to mean an ASPath starting with AS B followed by AS C, followed by path P_1 (which may contain zero or more AS's)

4 APPEND-TO-LAST-MOVE-AWAY(*update U*)

```
1  E = the last entry in the queue
2  while E is not the head of the queue  and
           E.NextHop != U.NextHop
4     E = the entry in the queue before E
5  if E is not the head of the queue
6     E = the entry in the queue after E
7  if E already has a pending ack that is an
           enabling acknowledgement for U
8     Acknowledge U
9  else
10    add U.Ack to the pending acknowledgements of E
```

Acknowledgement Messages

Updates are not performed to the data plane until acknowledgements are recived for the update messages that were previously sent out. PROCESS-LAST-ACKNOWLEDGEMENT outlines what an AS will do when the final acknowledgement is recieved for a particular update messages.

5 PROCESS-LAST-ACKNOWLEDGEMENT(*ack K*)

```
E = the queued update associated with K
Mark E as acknowledged
if E == head[Q]
   while the head of the queue is marked acknowledged
         E = the head of the queue
         Pop E off the queue
   Update the data plane to move to path E.ASPath
Send all acknowledgements in E.PendingAcks
```

First, the update is then marked as acknowledged. If the update is at the head of the queue, then all the acknowledged updates at the head of the queue are popped off, and the last update is actually executed on the data plane. All other updates are skipped, since they would just be over written by the following update anyway. Lastly, the AS will send all pending acknowledgements associated with that update. This will typically be only an acknowledgement to the update itself, but will occasionally include other acknowledgements that got queued by the process described in APPEND-TO-LAST-MOVE-AWAY.

C-BGP Proofs

No Routing Loops

LEMMA A.1. *In C-BGP, if Case 1 is followed for all updates, then when A acknowledges the last enabling acknowledgement for U, then either (i) or ((iia) and (iib)) is true:*

(i) A is not in U.ASPath

(iia) All updates to move to a path through U.NextHop were queued after U

(iib) A is not using a path through U.NextHop at the time of acknowledgment

PROOF. Acknowledgements are only sent for updates that are received, and there are four ways in which updates are processed. These four cases are outlined in PROCESS-BGP-UPDATE.

Case 1 In this case, there are no queued or current use of a path through *U.NextHop* and so (ii) is statisfied

Case 2 In this case, the process outlined in REMOVE-ALL-QUEUED-MOVES-TO is performed. This ensures that all entries moving to paths through *U.NextHop* are removed from the queue and so (iia) is satisfied. Additionally, this process outlines three cases:

Case 2a A is not currently a path through *U.NextHop* and so (iib) is satisfied

Case 2b Since A is in *U.ASPath* BGP will choose a path P through a different neighbor than *U.NextHop*. Additionally, since this update is at the head of the queue, the update will be performed before the acknowledgement is sent, as outlined in PROCESS-LAST-ACKNOWLEDGEMENT. Thus (iib) is satisfied.

Case 2c As outlined in APPEND-TO-LAST-MOVE-AWAY lines 7 through 9 the enabling acknowledgement will not be sent until the already queued move away from a path through *U.NextHop* is performed, and so (iib) is satified.

Thus in all sub-cases of Case 2, (iia) and (iib) are satified and condition (ii) is fully satisfied.

Case 3 and 4 In these cases, A is not in *U.ASPath* and so (i) is satisfied

Thus in all cases either (i) or (ii) are fully satified, and so the conditions of the lemma are always fully satified. □

THEOREM A.2. *C-BGP prevents forwarding loops if Case 1 is followed for all updates.*

PROOF. We'll prove this by contradiction. Without loss of generality, let's assume that a routing loop is formed at time t when A, transitions to use a path, B:P, announced by AS B, such that P goes through A. Let's also assume that at some earlier time $t_A < t$, A sends out the update, U, for it's transition to using a path through B. There are two possible ways for B to be using path P at time t.

Case 1: B is using P, or has queued a move to P, at time t_A and continues to use that path through time t. In this case, B must recieve update U since it's using a path, P, through A (or has queued a move to such a path), and continues to use that path until after U has been fully acknowledged. Thus, at some time $t_{B:Ack} > t_A$, B sends the enabling acknowledgement for U. However, at $t_{B:Ack}$, we know that B cannot be using path P, and cannot have queued a move to path P, because otherwise it could not have sent the enabling acknowledgement for U, as shown by lemma A.1. Additionally, since B is actually using a path through A, we know that A must recieve B's enabling acknowledgement before t, otherwise A cannot make the transition. Thus Case 1 cannot occur.

Case 2: B transitions to using P after t_A and before t, but has not queued an update to move to P before t_A. In order for B to make such a transition, it must receive the enabling acknowledgment from A, for an update sent after t_A, to move to using B:P. However, as shown by lemma A.1, A cannot send such an enabling acknowledgment while the move to a path through B is still in the queue, and so it cannot send such an acknowledgment before t. Thus Case 2 cannot occur.

Thus no update can form a routing loop, and so as long as the network begins in a loop free state, and all AS's follow the C-BGP protocol, it will remain in a loop free state. □

No Starvation

Next we'll show that an update cannot be slowed down by other updates happening cotemparaniously.

LEMMA A.3. *If an AS is ever indirectly waiting on itself, in order to send an acknowledgment or perform an update, then it should be able to immediately send the acknowledgment on which it is waiting.*

PROOF. An AS, A, can only wait on itself to perform an update or send an acknowledgment, if there exists a chain of dependencies such that A is waiting on a acknowledgment from X_1, who is waiting from an acknowledgment from $X_2 \dots X_N$, who is waiting on an acknowledgment from A. For any give chain of acknowledgment dependencies, let us break the AS's on that chain into three types, such that an AS, X_i is one of:

Pass-through AS: has no queued updates, but is currently using the path through X_{i-1} and will continue to use the path through X_{i-1} after the update.

From-path AS: is currently using the path through X_{i-1} but has queued an update to move to a path not through X_{i-1}

To-path AS: is not currently using a path through X_{i-1} but queued an an update to move to a path through X_{i-1} before it recieved the update.

Again, we'll prove this by contradiction. Let's assume that an AS is waiting on itself in order to perform an update and cannot immeadiately send the acknowledgement on which it is waiting. The dependency chain cannot have only pass-through AS's and to-path AS's, because that would mean an AS would have knowingly queued an update to move to a path that contains itself. Similarly, the dependency chain cannot have only pass-through AS's and from-path AS's, because that would mean that the current path has a routing loop in it. Thus the dependency chain must have both from-path and to-path AS's in it. Since the dependancies form a loop, this means there must be a to-path AS indirectly waiting on a from-path AS. If we call the AS making the to-path transition, X_t, and the AS making the from-path transition, X_f, then we know that X_t must have previously sent X_f an update indicating it's move to using the path through X_{t-1}, and that X_f has not acknowledged this update yet, otherwise there would be no dependancy here. Therefore, this acknowledgement is an enabling acknowledgement for any updates to X_t's path through X_{t-1}. So when X_t sends an update indicating a change in the path through X_{t-1}, X_f can acknowledge immeadiately since it already has pending an enabling acknowledgement. This is outlined in line 8 of APPEND-TO-LAST-MOVE-AWAY. Thus whenever such a loop exists at least one AS in the loop can acknowledge the update without waiting on any other AS, thus breaking the loop and ensuring an AS can never be stuck waiting on itself. □

THEOREM A.4. *As long as all AS's follow the rules, all updates are continuously making progress towards acknowledgment and no update can get starved.*

PROOF. We recognize the following facts:

- Given there are a finite number of AS's, and no AS indirectly waits on itself, there must be at least one AS that is not waiting on any other AS.
- There are a finite number of AS-neighbor connections.
- As outlined by the protocol descrition, no AS ever needs to acknowledge an update more than once per neighbor.

Thus, at any time at least one of a finite number of AS-neighbor connections can acknowledge the update, and so we conclude that all updates are continuously making progress towards completion, and no update can be starved. □

A.2 BGP-SIM

As described earlier, there are two possible classes of implementations of BGP-SIM: those that store the whole route in the packet itself, and those that store the route in the network. Since source routing is a well known concept, we assume that an implementation of BGP-SIM with the entire route stored in the packet, is straightforward. Thus we describe in detail here only the possible implementation of BGP-SIM where most of the forwarding information is stored in the network.

We'll describe this process with the assumption that when packets are sent between AS's, these packets can be marked in some way to indicate the *update incarnation* in which they originated. This marking can be done in many different ways, depending on the protocol used on the local network between the two AS's. It can be done through MPLS, IP-level tunneling, or even the use of virtual interfaces at the link-level. In the protocol description below we'll assume that packets can be *marked* with a *label* using one of these schemes.

A.2.1 *Protocol Description:*

When a AS would like to update the route that it is using to a particular destination, it performs the following steps:

1. It chooses a label that is not currently being used by any other route to the same destination (or if necessary, it garbage collects the route that has been installed the longest in order to use it's label)
2. It installs a route to the destination encoding four pieces of information:
   (i) the destination CIDR
   (ii) the label chosen in (1)
   (iii) the new next hop ip
   (iv) the new next hop label (from the BGP announcement)

   Thus when a packet arrives to be routed, it's route lookup is indexed by both (i) and (ii). It's label is changed to be the label from (iv) and it is routed to the next hop ip in (iii). Additionally, this new route is set as the default route for this destination in order to deal with packets with no label.
3. It sends a BGP update for the routing change including in it the packet label chosen in (1).
4. It sets a timer to time out the old route.

BGP-SIM Proofs

LEMMA A.5. *As long as the old route has not timed out on any router, any packet passing through an AS which has not yet updated it's data plane in response to the BGP update will be forwarded along the old path from that AS to the destination.*

PROOF. We'll prove this by induction.

Base Case: If an AS has not yet updated it's data plane, then it will continue to forward the packet to the same next hop IP address and the same label as the old path.

Inductive Step: Since all the old paths remain installed, if a packet is received with the old incoming label, at any node then it will be forwarded to the old next hop with the old outgoing label.

Thus, since the packet will follow exactly the same set of hops that it would have with the old path, it will exactly follow the old path. □

LEMMA A.6. *Any packet originating in an AS which has updated it's forwarding plane in response to the BGP* *update will be forwarded along the new path until it reaches an AS that has not received the update.*

PROOF. Again, we'll prove this by induction.

Base Case: If the AS originating the packet has already updated it's data plane, then it will generate a new packet that is sent with the label contained in the update it received.

Inductive Step: If the packet is received with the label that was published in the outgoing update, then the AS will send the packet with the label that was published in the incoming update that it received that caused it to publish the outgoing update.

Therefore each AS that has recieved the update will forward the packet along the path that is published in the BGP update and all packets will exactly follow the new path until they reach an AS that has not recieved the update. □

THEOREM A.7. *As long as the BGP update is processed by all AS's using this path before the ISP times out the path, plus the maximum packet delay in the network, there will be no routing loops:*

PROOF. The above lemmas show that a packet will entirely follow the new path until it reaches an AS that has not recieved the update and will then follow the old path to the destination. Thus, as long as neither the new or old path has a routing loop, there can be no routing loops. Note that a packet can pass through an AS twice, however the second time through it will have the label of the old path, and so it will be routed to the destination along the old path avoiding a loop. □