Network Working Group   P. Lapukhov
Internet-Draft   E. Nkposong
Intended status: Informational   Microsoft Corporation
Expires: March 06, 2014   September 02, 2013

Centralized Routing Control in BGP Networks Using Link-State Abstraction

draft-lapukhov-bgp-sdn-00

Abstract

Some operators deploy networks consisting of multiple BGP Autonomous-Systems (ASNs) under the same administrative control. There are also implementations which use only one routing protocol, namely BGP, as in [I-D.lapukhov-bgp-routing-large-dc], for example. In such designs, inter-AS traffic engineering is commonly implemented using BGP policies, by configuring multiple routers at the ASN boundaries. This distributed policy model is difficult to manage and scale due to its dependency on complex routing policies and the need to develop and maintain a model for per-prefix path preference signaling. One example of such models could be standard BGP community-based (see [RFC1997]) signaling, which requires careful documentation and consistent configuration. Furthermore, automating such policy configuration changes for the purpose of centralized management requires additional efforts and is dependent on a particular vendor's configuration management (CLI extensions, NetConf [RFC6241] etc).

This document proposes a method for inter-AS traffic engineering for use with the kind of deployment scenarios outlined above. No protocol changes or additional features are required to implement this method. The key to the proposed methodology is a new software entity called "BGP Controller" - a special purpose application that peers with all eBGP speakers in the managed network. This controller constructs live state of the underlying BGP ASN graph and presents multi-topology view of this graph via a simple API to third-party applications interested in performing network traffic engineering. An example application could be an operational tool used to drain traffic from network devices. In response to changes in the logical network topology proposed by these applications, the controller computes new routing tables, and pushes them down to the network devices via the established BGP sessions.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

BGP was intentionally designed as a path-vector protocol, since efficiently distributing link-state information for Internet-sized graph is virtually impossible. However, some network deployments leverage multiple BGP ASN to separate IGP domains, or simply use BGP as the only routing protocol. See, for example [I-D.lapukhov-bgp-routing-large-dc] which proposes using a BGP AS either per network device or "horizontal" device group, within a data-center. In such cases, the number of BGP ASNs is very small when compared to the Internet - on the order of few thousands in the largest case.

Under these assumptions, it becomes possible to build and maintain a link-state graph of the complete inter-AS topology and compute network paths based on this link-state information. In accomplishing this, it is desirable to avoid adding any protocol extensions so that current implementations can leverage the proposed method, such as those described, for example in [RWHITE2005]. Instead, this document proposes the use of a centralized agent (referred to as "BGP Controller" or simply "the controller") that peers with all eBGP speakers in the underlying network. The BGP Controller is responsible for constructing an up-to-date link-state view of the BGP inter-AS graph and pushing down routing information (prefixes and their associated next-hops) to the network devices via BGP updates. The new routing information reflects the results of link-state path computations performed by the controller. Such routing information push is possible because BGP supports the next-hop attribute that could be recursively resolved via either IGP or BGP. Notice that

while the controller pushes routing information to the device, the underlying BGP processes also compute the best-paths for the same prefixes using the path-vector logic in the regular way. However, the BGP Controller could override this information by manipulating BGP attributes of injected routes, such as LOCAL_PREF to make its own advertisements more preferred.

Third party applications can influence routing computations by creating logical alternations of the network link-state graph, e.g. changing the cost of the links from the BGP Controllers point of view. This document will refer to those constructs as "alternate topologies" (or simply "topologies" for short), while the original, unaltered, link-state graph will referred to as the "default topology". The controller would use these alternate topologies to make routing decisions different from those that BGP would have made based on available information. It is possible to create multiple alternate topologies and associate different prefixes with every topology, with the restriction that each prefix maps to one and only one topology. Once this mapping is defined, the BGP Controller would perform autonomously, detecting network faults and reacting by re-computing routing information as needed based on the effect that the failure has across all instantiated topologies.

In many aspects, the proposed method was inspired by and is similar to the "Routing Control Platform" [RCP], but differs in the fact that link-state discovery is done using BGP mechanics only, and overall BGP is the only protocol used to build the system.

2. Overview

2.1. Use Cases

Primary intended use case of the BGP Controller is inter-AS traffic engineering. This includes, but is not limited, to the following:

- Link/device overloading for the purpose of drying out traffic from a device. A link, or group of links, connecting one ASN to another could be declared as having "infinite" cost from the controller's viewpoint, causing the latter to re-compute paths and instruct the network devices to bypass those links. Notice that this does not include "internal" overload (inside an ASN), that may need to be done using IGP techniques.

- Traffic load-sharing among multiple links, e.g. links connecting two different ASN's. Multiple alternate topologies could be created where the same link is given different costs in each topology. These topologies will then have subsets of prefixes mapped to them, thus engineering different inter-AS paths for

these prefixes. Notice that for accurate load-sharing, knowledge of the traffic matrix may be required, but this requirement equally applies to any traffic engineering solutions. The load-sharing could be also accomplished using weighted Equal-Cost Multipath (ECMP), accounting for link capacities as "weights" to distribute different proportions of egress traffic to the peering points. See [KVALBEIN2007] for more information on the multi-topology techniques in general and [I-D.ietf-idr-link-bandwidth] for information on weighted ECMP signaling in BGP.

The main benefit of the proposed approach is centralized control of the above functions. There is no need to configure policies on multiple devices, as all routing changes could be done using the uniform light-weight API to the controller. This ensures ease of automation and consistent changes. Furthermore, such a centralized model should be deployed to augment the classical distributed routing policy configuration. The advantage is that centralized control could be disabled at any time, falling the network back to the "traditional" BGP decision model, thus allowing for a safe state to roll-back to. Next, knowing the link-state of the network may allow avoiding the BGP path-hunting problem, and improve global BGP convergence timing in a large group of heavily meshed ASNs. Additionally, to avoid the phenomena of routing micro-loops the controller could enforce certain ordering for the network device programming sequence. Specifically, every time a link-state change is proposed to the controller, the devices in the network are programmed starting with those farther away from the change in terms of the metric of the existing graph. The same logic applies to link-down conditions detected by the controller via the health probing mechanism described below.

2.2. Architectural Assumptions

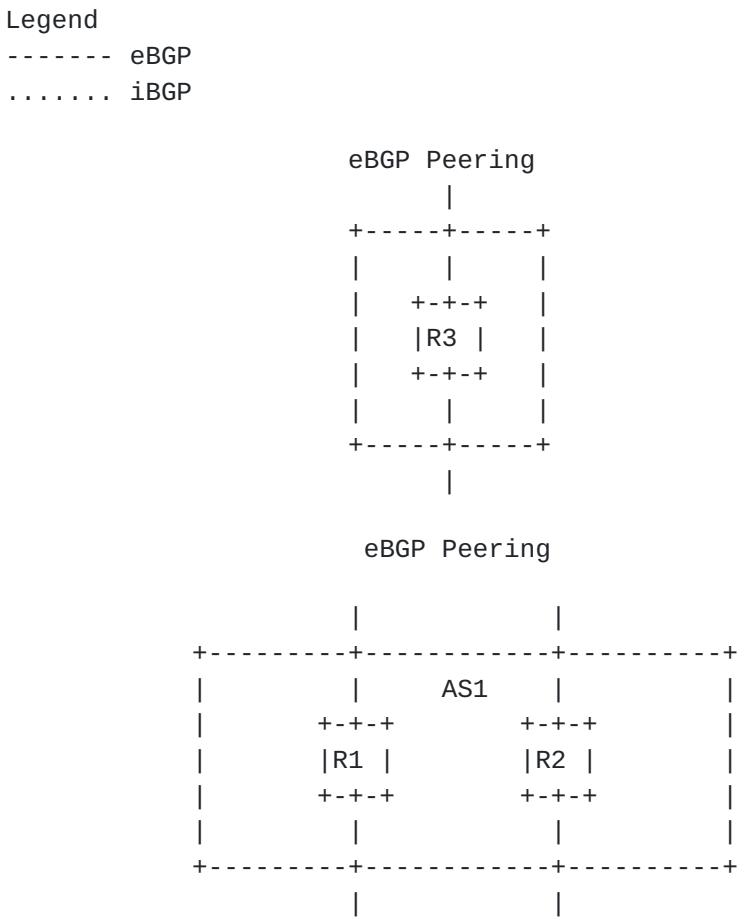
Firstly, the devices in the network are assumed (but not required) to have minimal BGP policy applied, enough for them to exchange routing information and compute best-paths based on shortest AS_PATH lengths. This means that the configured policy should not override best-path selection process using LOCAL_PREF or any other BGP attributes for enforcing a custom routing policy. The assumption of the "minimal policy" allows for making the BGP Controllers update logic less intrusive, as described further in the section Section 4.7. Next, every device is assumed to advertise a locally bound prefix into BGP for the purpose of BGP peering with the controller. That is, the controller peers "inband" with the devices it controls - either by initiating iBGP sessions to all devices or by passively accepting the sessions from the devices. As will be shown in the Section 5, inband peering requirement is important to avoid inconsistencies between multiple controllers programming the same network.

Another major assumption is how the link-state graph vertices are defined. From the BGP Controller perspective, there are two type of vertices:

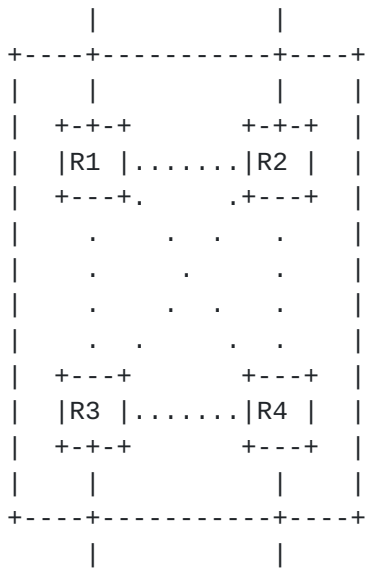
- Type 1, Individual Devices: BGP Speaker(s) that have the SAME BGP ASN configured, with the restriction that none of these speakers peers with each other, inside this ASN. This could be a single speaker in its own ASN as well. Each of these speakers is treated as a vertex on its own. Peering with other ASN's is not restricted. Notice how this is different from the traditional notion of BGP ASN, where all speakers are assumed to be part of the same iBGP mesh.

- Type 2, Complete BGP ASN: BGP Speakers in the SAME BGP ASN with the normal requirement that they ALL exchange their BGP views via iBGP, using either full-mesh or any other approach for full internal BGP state synchronization. All of these BGP speakers are grouped into a single graph vertex.

The following Figure 1 illustrates this concept:



Type 1: Each device is individual graph vertex
(three vertices, each with two edges).



Type 2: All devices below are grouped into
single vertex with four edges.

Figure 1: Graph Vertices

Routing information could be associated with a graph vertex either by means of static binding or dynamic discovery: this process is described in details in sections Section 4.3. When programming the network prefixes into the devices, the controller does not inject a prefix back in the vertex the prefix is associated with.

The BGP Controller decision logic is independent of the address family, and could apply to both IPv4 and IPv6 prefixes equally. It is possible to run two independent controllers, one for each address family. This allows for full "fate decoupling" between the address families, though may result in duplication of the link state information.

The edges of the constructed link-state graph may have two attributes: metric, which is additive, and capacity (bandwidth) that is non-additive. The former is used to compute shortest paths, and the latter could be used to compute ECMP weight values in case where multiple equal-cost paths exist to the same vertex. For every ECMP path, the minimum capacity value that occurs along that path will be used as its weight by the controller, if the underlying network supports weighted ECMP functionality.

2.3. BGP Controller

The Figure 2 demonstrates the BGP Controller peering with the network devices. Multiple managed devices peer via eBGP following the traditional BGP design. For simplicity, we assume that every device belongs to it's own ASN - see Section 4.6 for more information on handling the "compound" Type-2 vertices consisting of multiple BGP speakers interconnected with iBGP mesh. Prefixes P3, P4 and P5 are associated with the devices (vertices) in ASNs 3, 4, and 5 respectively using techniques described in Section 4.3. The other remaining vertices are assumed to be purely transit for the purpose of this discussion.

These devices exchange routing information in the usual manner and the BGP Controller establishes iBGP peering sessions with every device. It uses the technique described in section Section 3.1 to build the inter-AS link-state graph. For now, it is sufficient to say that the discovery process uses special "beacon" prefixes dynamically injected into the network and relayed back to the controller to discover the state of the links interconnecting the graph vertices.

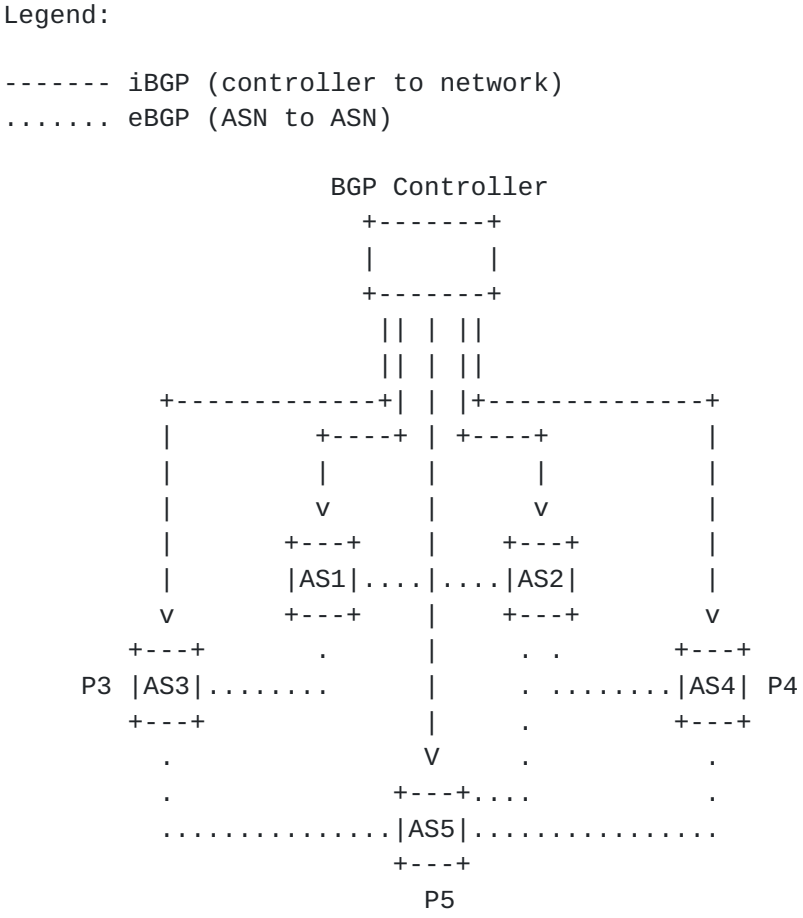


Figure 2: BGP Controller

At this point, the BGP Controller has knowledge of the link-state graph as well as the prefixes associated with every vertex, and can now run Dijkstra's SPF algorithm to compute shortest paths between vertices. A result of this computation would be routing tables built for every vertex. The Section 3.2 below demonstrates the adjacency list built by the controller for the above topology, as well as routing-tables computed for every vertex. The next-hops in the routing tables presented in the figure are simply the vertices to send the packets to. When programming the network devices, the actual IP addresses of the next-hops are computed as described in Section 4.1 section. This routing state corresponds to the unaltered (default) topology.

3. Link-State Abstraction and Multiple Topologies

This section provides detailed information on the link-state abstractions used by the controller and how those are used to perform traffic engineering in the underlying network.

3.1. Link-State Discovery Process

The network devices that the controller peers with establish eBGP peering sessions with each other. The fact that there is one-to-one correspondence between eBGP sessions and underlying IP link allows using the state of the eBGP session as the indication of the IP link health. Specifically, this is accomplished by injecting special "beacon" prefixes into every vertex (which could be a device or collection of devices interconnected with iBGP mesh) and expecting those beacons to be re-advetised back to the controller by every vertex adjacent to the point of injection. If a particular BGP session is down, the injected prefix will not be re-advertised by the affected peer back to the controller, allowing us to conclude that the corresponding link is down.

The Figure 3 demonstrates this process. For simplicity, we assume that every device belongs to its own BGP ASN. The BGP controller injects prefix X into device R1 and expects to hear this prefix from device R2. At the same time, it is desirable to prevent this prefix from leaking any farther than one hop away from R1, i.e. make sure it is not re-advertised to R3. To accomplish this, prefix X could be tagged with a special community value, which is replaced with the well-known community "no-export" when advertising over eBGP session. Because of this policy, the prefix will be announced back to the controller as it uses iBGP session for peering, but not any further to eBGP peers of router R2 in our case. An alternative to using the standard BGP communities could be leveraging the wide-communities

limiting the scope of the announced prefixes - see [I-D.raszuk-wide-bgp-communities] for more details on this technique.

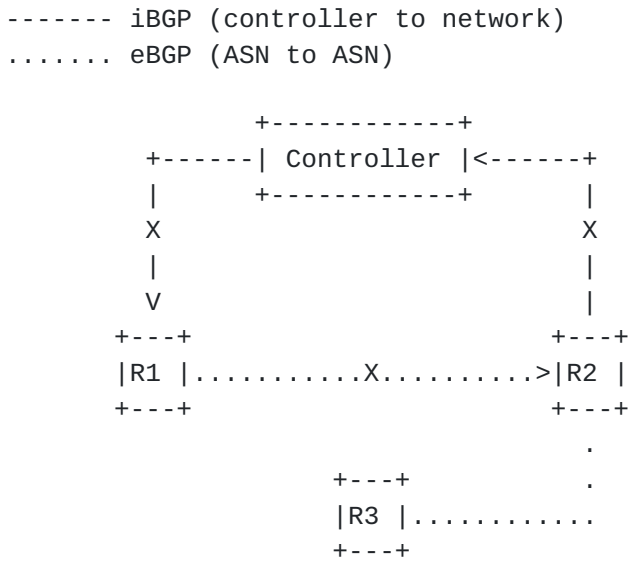
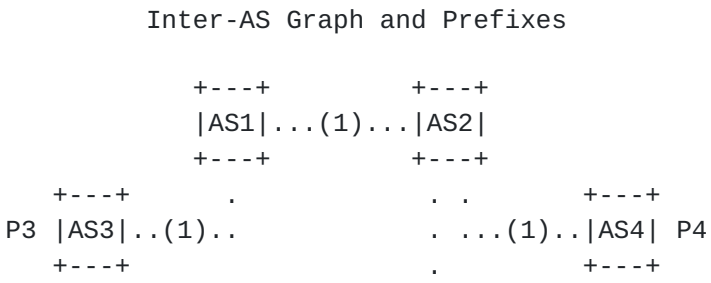


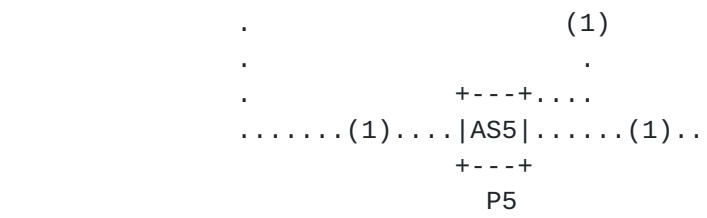
Figure 3: Link-State Discovery

Using this technique, the controller is able to build a view of the links connecting the graph vertices. Notice that if two parallel links connect vertices, this method will not be able to differentiate between them. For simplicity, the proposal is that such parallel links should be grouped into a single logical IP link using, for example, [IEEE8023AD] technology.

3.2. The Default Topology

When the controller starts, it discovers the current network graph and computes the routing table assuming that all links have the same metric value. The Figure 4 illustrates the adjacency list describing the graph taken from Figure 2 along with the routing table computed for every vertex/ASN. The numbers on the graph edges designate the link costs.





Inter-AS Graph Adjacency List

Src	Dst ASNs
AS1	AS2,AS3
AS2	AS1,AS4,AS5
AS3	AS1,AS5
AS4	AS2,AS5
AS5	AS4,AS2,AS3

Per-ASN Routing Table

AS	Prefix:Next-Hop(s)
AS1	P3:AS3,P4:AS2,P5:[AS2,AS3]
AS2	P3:AS1,P4:AS4,P5:AS5
AS3	P3:Self,P4:AS5,P5:AS5
AS4	P3:AS5,P4:Self,P5:AS5
AS5	P3:AS3,P4:AS4,P5:Self

Figure 4: Unaltered Routing State

3.3. Alternate Topologies

Assume the following TE requirements for illustrative purposes:

- Traffic from AS4 to P5 needs to traverse AS2.
- Traffic to P4 from AS5 needs to ECMP over two paths: direct and via AS2.
- Traffic from AS3 to P5 must not use the direct path.

These requirements could be satisfied with two different topologies:

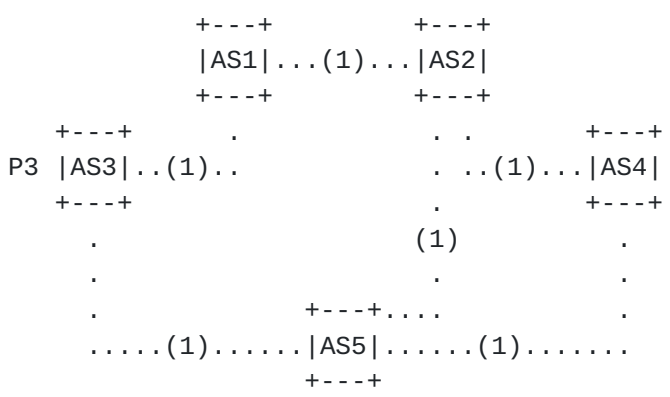
- Topology 1 has "very large" metric assigned to the links between AS4,AS5 and AS3,AS5.

- Topology 2 has metric value of 2 assigned to the link between AS4 and AS5.

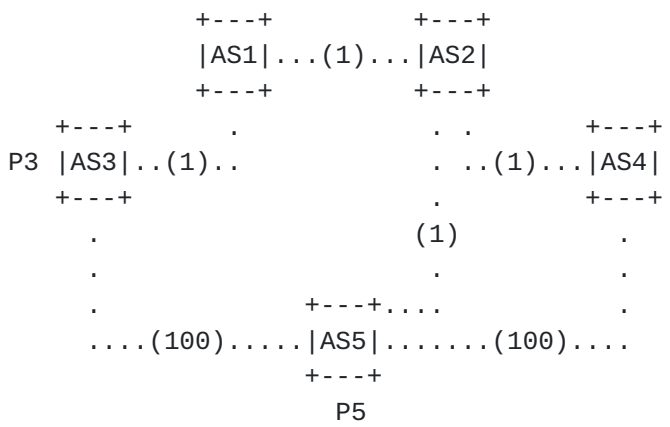
The prefixes map to the topologies as following: P5->Topology1 and P4->Topology2. P3 should retain mapping to the default (unaltered) topology, which we would refer to as Topology 0 to refer to all topologies by their numbers. The assumption of "very large" metric

is important - the path containing this link could still be used if all alternate paths are down because of physical failures. For simplicity, we assume "very large" equals to 100 in the case under consideration. The set of topologies and associated prefixes would look as on Figure 5, where numbers on the links designate their metrics.

[Topology 0]



[Topology 1]



[Topology 2]

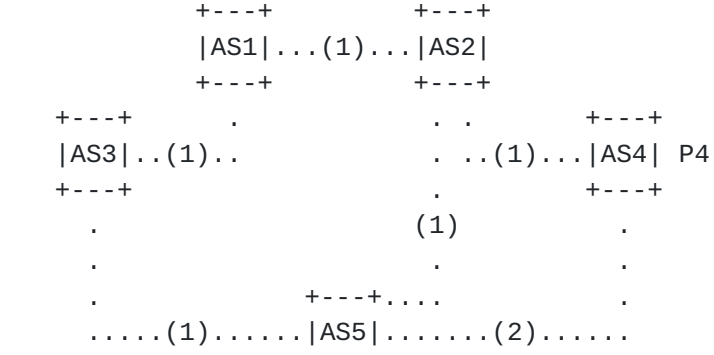


Figure 5: Alternate Topologies

Based on the set of topologies presented above, the BGP Controller will compute the routing tables shown in Figure 6, which reflects the desired traffic engineering goals defined previously. The entries that differ from the routing decisions in unaltered topology are highlighted with the asterisk (*) characters. Notice that AS3 now sees P4 as ECMP reachable via AS1 and AS5, because of the metric change in Topology 2. The original traffic engineering policy requirements did not call for that, but this result appears because of the change made between AS4 and AS5, which is a natural effect with shortest-path, destination-based forwarding techniques.

Per-ASN Routing Table

AS	Prefix:Next-Hop(s)
AS1	P3:AS3,P4:AS2,*P5:AS2*
AS2	P3:AS1,P4:AS4,P5:AS5
AS3	P3:Self,*P4:[AS5,AS1]*,P5:AS1
AS4	P3:AS5,P4:Self,*P5:AS2*
AS5	P3:AS3,P4:*[AS4,AS2]*,P5:Self

Figure 6: Multi-Topology Routing Tables

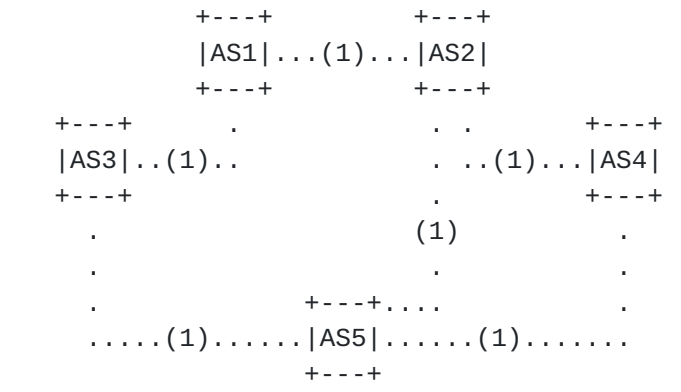
The controller will push the computed routing tables to the network devices using higher LOCAL_PREF values to ensure that the new information overrides the routing decision that "traditional" BGP processes running on the BGP speakers have already made. It is possible to use other attributes to signal better preference, but LOCAL_PREF has the benefit of being used very early in the BGP tie-breaking process.

3.4. Overloading a Vertex

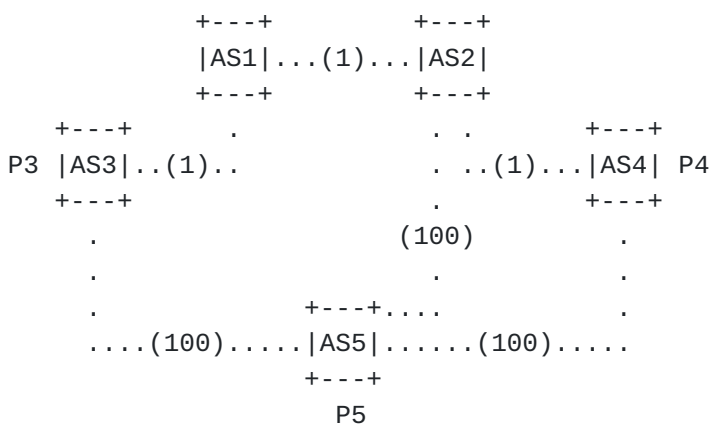
This section illustrates a special, but important practical case of "overloading" a graph vertex, such that all traffic bypasses the vertex. This operation could be used in a scenario in which a particular network device needs an upgrade and requires all traffic to be dried out of it. The Figure 7 demonstrates the implementation

of this policy with respect to the AS5 vertex. The Topology-0 has no prefixes mapped to it, but all prefixes are mapped to Topology-2 instead. This topology has the cost of 100 assigned to all links connected to AS5, which forces all traffic to avoid transiting AS5.

[Topology 0]



[Topology 2]



Per-ASN Routing Table

AS	Prefix:Next-Hop(s)
AS1	P3:AS3,P4:AS2,*P5:[AS2,AS3]*
AS2	P3:AS1,P4:AS4,P5:AS5
AS3	P3:Self,*P4:AS1*,P5:AS5
AS4	P3:*AS2*,P4:Self,P5:AS5
AS5	P3:AS3,P4:AS4,P5:Self

Figure 7: Overloading a Vertex

4. Implementation Details

4.1. Programming Next-Hops

As mentioned previously, the prefixes that the controller injects in the network needs to have their next-hops properly resolved. In the simplest case, the next-hops could be the remote IP addresses of the links directly connected to the device programmed by the controller. This, however, adds certain complexities due to the IP address variability on the point-to-point links connecting the network devices. An alternative could be injecting pre-generated next-hops into the devices - one per device - and resolving them recursively via BGP.

Specifically, every graph vertex would have a host route (either IPv4 or IPv6) associated with it. The controller would inject this prefix into the respective device(s) (see Section 4.6 associated with this vertex, tagged with the special community value discussed in the section Section 3.1. Moreover, for simplicity, it is possible to re-use the same prefix used for link-state discovery as the value of the next-hop attribute, thus reducing the amount of supplementary routing state injected by the controller.

Next, it is easy to notice that using the special BGP community to limit the beacon/next-hop prefix propagation is not strictly necessary. Indeed, the controller may simply discard all "special" prefixes whose AS_PATH contains more than one AS-hop. However, this will result in unneeded routing state propagated in the network, which is not desirable from manageability perspective.

4.2. Equal-Cost Multipath Routing

In many practical topologies, the controller may find multiple equal-cost paths from one vertext to another. It may then proceed programming multiple paths for the prefixes affected by this decision. Either of the two ways could accomplish the multiple-paths programming requirement:

- Using the BGP Add-Path extension, [I-D.ietf-idr-add-paths] specifying multiple next-hops values.

- Using the Diverse Path Advertisement method presented in [RFC6774] to inject multiple paths.

Furthermore, it is possible to implement weighted ECMP functionality with this approach, relying on [I-D.ietf-idr-link-bandwidth] for weight signaling. The graph edges could have weights associated with them, and a given path's weight computed as the minimum weight value along the path, as mentioned previously. The logic behind the weight selection is outside the scope of this document.

4.3. Prefix Discovery Process

In order to build routing state information, the controller needs to know the "leaf" prefixes associated with the graph vertices. There are two ways of accomplishing this: either defining a static mapping of prefixes to vertices in the BGP controller configuration, or by letting the controller learn those prefixes in dynamic fashion. In both cases, the assumption is that the network reachability information is already advertised into BGP, such that regular "in-band" routing model is working.

The controller may dynamically associate a prefix with a vertex by using two properties: firstly, by observing an empty AS_PATH in the prefix received from the managed device. Secondly, by filtering out prefixes injected for the purpose of network health discovery and next-hop programming. The controller treats everything that matches these two criteria as the routing information associated with the respective vertex.

4.4. Sequenced Device Programming

Distributed routing systems are susceptible to transient inconsistencies when a network state changes in such a way that requires changing the best-paths election. Since a topological event (e.g. a link flap) is not propagated in an instant, devices that are closer to the origin of the event would update their forwarding tables faster, as compared to others. The devices directly adjacent to those that have their tables already updated would still be using old forwarding state. This would create transient routing loops for the time it takes to fully synchronize the forwarding state of all devices.

Since the controller is aware of the full network topology, it may avoid the above scenario by pushing the routing updates in proper sequence - starting with the vertices that are farthest away from the location of the event. This way the newly programmed state will "implode" toward the change, as opposing to "exploding" from the events point of occurrence. Such sequencing is similar to the process outlined in [RFC6976], but relies on centralized programming, which makes it very simple to implement.

4.5. Mapping Prefixes to Topologies

The controller needs a manageable way of associating discovered prefixes with any of the topologies defined by the third-party applications. As mentioned previously, all prefixes are by default mapped to the default topology, which corresponds to the actual network state. Once an alternate topology has been defined, prefixes could be mapped to this new topology. One possible way of implementing such mapping table could be by maintaining a radix tree data-structure, which associates a prefix with the corresponding topology. Using longest-match lookup in this table for each discovered prefix would then yield the topology that this prefix belongs to. This allows for easy and natural grouping of prefix-to-topology mappings, while maintaining familiar semantics of longest-match routing lookups. To implement the default mapping, the prefixes 0.0.0.0/0 and ::/128 should always be in the radix tree, pointing to one of the defined topologies. When those prefixes are deleted per application request, the BGP controller would need to re-insert them, linking back to default topology again.

4.6. Autonomous Systems with iBGP Peering Mesh

The BGP Controller treats BGP ASN's that have a form of internal BGP mesh differently than systems that do not peer over iBGP. Such systems are perceived as an atomic opaque graph vertex for the purpose of next-hop and beacon prefix injection. The routing inside such ASN is not defined by the controller, but rather relies on some other mechanism, such as IGP. The controller only defines egress points out of the ASN, and possibly can specify weights associated with exit points, to allow for weighted ECMP load-distribution. This treatment naturally arises from the fact that iBGP injected beacon prefixes are not relayed to iBGP peers. Furthermore, the beacon prefixes learned from eBGP neighbors are propagated to all iBGP peers, but not relayed back to the BGP Controller when learned over iBGP session. Thus, the controller will discover peering links of every "edge" router in such BGP ASN with all external peers, but will not be able to see the internal iBGP peering mesh.

If the underlying ASN implements iBGP route reflection or BGP Confederations, only the routers that form eBGP sessions with external ASN's need to have the routing information injected into them. The routing information will disseminate to the internal speakers by means of normal BGP replication process, with unmodified next-hops and LOCAL_PREF attribute value, thus ensuring that it overrides the normal "in-band" routing information.

When programming ECMP paths, it may happen so that the egress points specified by the controller do not satisfy iBGP requirements for

multipath (e.g. IGP costs to reach the egress points could be different). In such case, normal BGP tie breaking will occur and only ECMP-equivalent paths will be installed in the RIB. Alternatively, if the underlying ASN implement tunneling techniques, it is possible to perform load sharing even if the IGP costs toward the BGP next-hops are different.

4.7. Minimizing Controller-Injected State

The BGP Controller can push down all of the prefixes it computes paths for: that is, all prefixes known in the network. This means that for every prefix present in the "regular" eBGP interconnected topology the controller will inject the same prefix with different attributes. It is also possible for the controller to push down only the "delta" between the prefixes that need their next-hops/paths changed, based on the supplied policy. This mode of operation requires that the underlying network finds the best-paths between the graph vertices using the "shortest-path logic", where the path length equals the length of the AS_PATH attribute. This is equivalent to running Dijkstra's SPF algorithm on graph unit metric values assigned to the edges. This is needed since the controller performs path computation using SPF logic, and BGP could elect different paths if some policies are present. Ensuring that both the underlying network and the controller perform the same computations effectively allows for the "delta" mode operations.

Publishing only the "delta" state to the network means more "intelligent" work on the controller side and special requirements to the network policies. However, the benefit is significantly reduced intervention in the regular forwarding since majority of the state is not likely to change in many cases. Once again, it is possible to implement the mode where the controller overrides all routing information.

5. Handling Failure Scenarios

This section reviews two different type of failure scenarios: failures in the underlying network and the controller failures.

5.1. Underlying Network Failures

Either vertex (if it's a device) or graph edge (network link) may fail. For the BGP Controller, underlying failure be it edge or vertex, is visible only after all eBGP session interconnecting two vertices have failed. This could be driven either by an event, such as link down condition, which is typically fast, or by BGP keepalive timer expiration, which is naturally slower. When this happens, the BGP processes withdraw the corresponding beacon prefixes and the

controller will declare the corresponding edge down. This will result in re-run of SPF for all active topologies and push of new routing information down to the network. Since the central controller is involved in reconvergence, the restoration time will be longer, compared to the restoration process driven purely by underlying BGP processes. Indeed, the restoration time now include failure detection time, SPF re-computations and new prefixes push. However, it could be observed that such centralized reconvergence is free from the BGP Path-Hunting problem, and hence improvements could be noticed in complex meshed topologies.

Furthermore, recovery could be faster if multiple paths (ECMP) exist for a prefix, and only a single path fails. In this case, BGP process will simply invalidate the failed path even before the controller has signaled removal, and will continue with using only the active paths. The details of this reconvergence are complicated, as changing ECMP is a hardware dependent operation. Furthermore, some implementations may support the "consistent hashing" technique that minimizes impact of ECMP group base size change on flow affinities, as described in [RFC2992].

5.2. BGP Controller failures

Under normal circumstances, an operator may shut down a controller for maintenance or other reasons. In this case, it is expected that BGP sessions be closed following normal BGP process, that is sending a BGP Notification message and terminating the TCP session. As a result, all routers will withdraw the prefixes injected by the controller and recalculate the best-path.

If the controller fails abnormally, e.g. process crashes, the TCP sessions that connect it to the underlying devices either will be torn down, or be closed upon expiration of BGP keepalive timer. The latter will cause some delay before prefixes announced by the deceased controller are withdrawn. For the duration of that time, the network will be forwarding traffic using possibly stale information. Link/device failures will be handled locally, and in some cases may cause traffic black-holes, if the only programmed path fails. The duration of this "state" time is equal to the time it takes to detect the controller failure, and update the BGP LocRIB, followed by RIB/FIB reprogramming.

It is possible to use a single BGP controller along with BGP routing persistence feature, to maintain the injected paths even after the BGP Controller failure (see [I-D.uttaro-idr-bgp-persistence]). After the controller restarts, it will simply refresh the "stale" routing information. In this scenario, forcing the network to revert to the traditional BGP-based routing could be accomplished by instructing

the controller to inject its paths with low LOCAL_PREF value, less than the default used in the network. The possible risk is that the controller may fail in such a fashion that it will not be able to inject any information in the network.

5.3. Multiple BGP Controllers

If a single BGP Controller is present in the network that does not implement BGP route persistence, the controller failure would result in the network becoming unmanaged, and falling back to traditional BGP routing. To maintain resilience, it is possible to run multiple parallel BGP Controllers, assuming that they supply the network with the same routing information, and differentiate themselves as 'primary' and 'backup'. The latter property could be accomplished by using different LOCAL_PREF attribute values for primary/secondary controllers - this allows having multiple controllers, backing up each other.

With multiple BGP Controllers, it becomes critical for all of them to perform the same routing decisions. Even though only one controller is programming the network, the backup paths injected by the others must be consistent with the primary. To accomplish that, all controllers must:

- Have the same view of the underlying network topology - i.e. build the same link-state graph. In the simplest case, this could be accomplished by relying on eventual consistency, that is assuming that under non-partitioned scenario the controllers will eventually receive the same link-state probe prefixes and build the same resulting link-state database. Alternatively, a consensus protocol, e.g. [PAXOS] could be executed amongst the members of the redundant group to synchronize the link-state database of the master process with the secondary processes. This would ensure strong consistency of the link-state database, but could be over-bearing in terms of the state that may need to be kept replicated reliably.

- Maintain the same topology definition database and prefix-to-topology mapping table - as commanded by external applications. This is similar to the previous approach, but would involve much less state to synchronize. Specifically, the topology definitions (e.g. new link costs) and prefix to topology mapping information need to be distributed. This state is submitted to the controllers via an API defined for the third party applications. As before, it could be assumed a responsibility of an external application to program all controllers with the same state and ensure consistency. Alternatively, another strongly consistent database could be used, leveraging the same consensus protocol.

5.4. Network Partitioning

This section reviews the possible "partitioning" scenarios, where parts of the network may become managed by different controllers. Situations like this are possible if the controllers are deployed diversely, and may end up in situation where one or more of the controllers lose iBGP peering sessions with some network devices. The main concern in such situations is programming the devices with inconsistent information that may cause routing loops.

Firstly, notice that if device A can learn the "peering source" prefix announced by device B, and the BGP Controller can peer with A, then by transitivity the controller can also peer with B. This means that either the controller and device A cannot learn any routing information from B, or both of them can - excluding transient situations. This property ensures that under proper configuration a set of devices is either completely managed, or completely unmanaged - that is, they share the same fate. This eliminates the scenario where device A is programmed by the controller X, device B is programmed by the controller Y and the devices can each each other inband.

Secondly, for the transient cases, when A and B have in-band connectivity, but for some time A is programmed by X and B is programmed by Y. Recall that absence of the iBGP session to the device translates into the fact that this device is declared as having "infinite" costs in the link-state database. Thus, X will always bypass B and Y will always bypass A, and hence a routing loop may never form between A and B.

6. Controller API

This section provides a set of requirements and guidance to the BGP Controller API. The general recommendation is to base the API on stateless principles, such as found in [REST] model. This approach is efficient since no real-time event passing between the controller and third-party application is needed, e.g. for the purpose of active reaction to network failure events. The proposed controller model assumes those events are handled by the messages exchange in the network-controller loop. The following sections are structured the around "CRUD" - Create, Read, Update, Delete operations commonly used in REST model and use the HTTP verbs and pathnames for illustration. Furthermore, applications will be referenced as clients and the BGP Controller as the server in the text below interchangeably, though the API could be implemented by a module separate from the main BGP Controller logic.

6.1. Pathnames and document names

The server presents the following pathnames to group various objects:

- "/lsdb" - This is the document that stores the currently discovered inter-AS graph link state (link-state database). This document cannot be modified, only read. The LSDB data structure is a graph, represented in one of the common formats - e.g. as two collections: vertices and edges, where edges have associated states and weight (capacity).

- "/topologies/" - This is a directory that stores documents corresponding to different topologies. Every document contains a topology definition.

- "/mappings/ipv4" - This is the document that stores the IPv4 mappings to the topologies. Notice that if the 0.0.0.0/0 prefix is not found it this file, it is implicitly mapped to the default topology. Internally in the BGP Controller this is stored as an efficient radix-tree, but the document represents the mappings as a collection of prefixes and associated topologies.

- "/mappings/ipv6" - This is the document that stores the IPv6 prefix mappings to the topologies. Same as IPv4 mappings, with except to different address family. As with the IPv4 case, if the ::/0 prefix is not found in this document, it is implicitly mapped to the default topology.

6.2. Encoding of the documents and objects

Either JSON or XML is an acceptable format for encoding the document contents for programmability. JSON is preferred due to its lightweight nature and simpler semantics for transporting data structures. The documents passed with RESTful calls will contain logical descriptions of the graph vertices and edges. A vertex is uniquely identified by an opaque name, e.g. a text string. The mapping between this identifier and the underlying network devices is to be done elsewhere in the controller data structures, and does not need to be exposed to the applications.

6.3. Creating & Deleting State

The only state that could be created is the collection of topology definitions, under the "/topology/" directory. The topology objects are to be created using the "POST" HTTP operation - supplying some basic content, e.g. empty set of the links and associated costs using the appropriate encoding. Correspondingly, a topology could be deleted using the DETELE operation. Notice that the default topology

is not present in this directory, and thus could never be deleted. Notice that the separate "mapping" documents will be referencing the topology names, and when a topology is delete such mapping will become invalid. It up to the implementation to handle such referential integrity - e.g. by ignoring such entries in the mapping document, or disallowing the topology file to be deleted as long as active references are present.

6.4. Reading State

Every document described above could be read and transported to the client using the HTTP GET request. The document is transported completely in the corresponding encoding. It is up to the controller to implement proper read/write locking to avoid inconsistencies in data when multiple clients are present. No locking API should be ever exposed to the client, since that would affect the stateless nature of the communications. Notice that reading the link-state database is mostly informative to the client, since handling of the network failures is performed by the BGP Controller.

6.5. Writing State

The topology definition documents and the IPv4/IPv6 mapping tables could be fully re-written using the HTTP PUT verb. This means that with every operation, the client must supply the full new document, not an incremental change. It's up to the client to perform the merge of the new change with the already existing information. If consistency across multiple writers is required, it should be implemented by the clients, possibly via the use of an external shared locking API. Referential integrity checks could be implemented in the controller, e.g. to validate that the topology references in the mapping actually exists, or alternatively could be left to the client.

It is possible to implement incremental changes using the HTTP PATCH verb semantics (see [RFC5789]) in the server. In this case, it's up to the server to perform proper merge of the incremental change and ensure there is no conflicts or duplicates. This is a more complex model as compared to the simple "PUT" logic.

6.6. Typical API Call Sequence

A typical sequence of actions for a client willing to perform traffic engineering could be as following (assuming absence of the PATCH operation):

- Decide which prefixes are to be affected by this operation.

- Create a topology to perform the link-state operation, or re-use the one previously created by this application. Verify topology existence using the GET operation in the "/topologies" directory.

- Add new links with the desired costs to the topology. If the topology alredy exists, read it first using GET operation, and then perform merge on the client side, later submitting the updated topology using PUT operation.

- Obtain current prefix mappings for the desired address family using the GET operation. Parse the mappings and perform any consistency checks required, followed by adding the entries for prefixes to act upon, mapping them to the topology created/updated above.

- HTTP PUT the new mappings file, updating the one that existing in the server as a whole.

6.7. Limitations

The API is purposely focused only on routing information manipulation, and does not provide any ways to verify the requested operation has been accomplished. Such monitoring should be done separately, using either mechanics available in BGP (e.g. by learning of the prefixes' new paths via separate session) or outside of BGP, e.g. in BGP Monitoring Protocol ([I-D.ietf-grow-bmp]) or Multi-Threaded Routing Toolkit ([RFC6396]).

7. Security Considerations

The design of the BGP Controller in its simplest form assumes no access control in the API is presents to the third-party applications. Access could be limited at the transport level, e.g. by using protocol (HTTP) authentication or access control capabilities, but the API itself does not provide any logic to segregate applications - i.e. there is currently no way to limit an application to manipulating only a certain subset of the IP address space.

8. Acknowledgements

The authors would like to thank Robert Raszuk for reviewing the document and providing valueable feedback.

9. References

9.1. Normative References

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC1997] Chandrasekeran, R., Traina, P., and T. Li, "BGP Communities Attribute", RFC 1997, August 1996.

9.2. Informative References

[I-D.lapukhov-bgp-routing-large-dc]
Lapukhov, P., Premji, A., and J. Mitchell, "Use of BGP for routing in large-scale data centers", draft-lapukhov-bgp-routing-large-dc-06 (work in progress), August 2013.

[I-D.ietf-grow-bmp]
Scudder, J., Fernando, R., and S. Stuart, "BGP Monitoring Protocol", draft-ietf-grow-bmp-07 (work in progress), October 2012.

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, December 2006.

[RFC6774] Raszuk, R., Fernando, R., Patel, K., McPherson, D., and K. Kumaki, "Distribution of Diverse BGP Paths", RFC 6774, November 2012.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C., Francois, P., and O. Bonaventure, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach", RFC 6976, July 2013.

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, November 2000.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6396] Blunk, L., Karir, M., and C. Labovitz, "Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format", RFC 6396, October 2011.

[I-D.ietf-idr-add-paths]
Walton, D., Retana, A., Chen, E., and J. Scudder, "Advertisement of Multiple Paths in BGP", draft-ietf-idr-add-paths-08 (work in progress), December 2012.

[I-D.ietf-idr-link-bandwidth]
Mohapatra, P. and R. Fernando, "BGP Link Bandwidth Extended Community", draft-ietf-idr-link-bandwidth-06 (work in progress), January 2013.

[I-D.raszuk-wide-bgp-communities]
Raszuk, R., Haas, J., Amante, S., Steenbergen, R., Decraene, B., and P. Jakma, "Wide BGP Communities Attribute", draft-raszuk-wide-bgp-communities-03 (work in progress), July 2012.

[I-D.uttaro-idr-bgp-persistence]
Uttaro, J., Chen, E., Decraene, B., and J. Scudder, "Support for Long-lived BGP Graceful Restart", draft-uttaro-idr-bgp-persistence-02 (work in progress), July 2013.

[JAKMA2008]
Jakma, P., "BGP Path Hunting", 2008, <https://blogs.oracle.com/paulj/entry/bgp_path_hunting>.

[PAXOS] Wikipedia, ., "Paxos", , <http://en.wikipedia.org/wiki/Paxos_(computer_science)>.

[REST] Wikipedia, ., "Representational state transfer", , <http://en.wikipedia.org/wiki/Representational_state_transfer>.

[RWHITE2005]
White, R., "Graph Overlays on Path Vector: A Possible Next Step in BGP", June 2005, <http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_8-2/graph_overlays.html>.

[KVALBEIN2007]
Kvalbein, A. and O. Lysne, "How can Multi-Topology Routing be used for Intradomain Traffic Engineering?", 2007.

[IEEE8023AD]
IEEE 802.3ad, ., "IEEE Standard for Link aggregation for parallel links", October 2000.

[RCP] Caesar, M., Caldwell, D., Feamster, N., and J. Rexford, "Design and Implementation of a Routing Control Platform ", March 2005, <http://www.cs.princeton.edu/~jrex/papers/rcp-nsdi.pdf>.

Authors' Addresses

Petr Lapukhov
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Phone: +1 425 7032723
Email: petrlapu@microsoft.com
URI: http://microsoft.com/

Edet Nkposong
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Phone: +1 425 7071045
Email: edetn@microsoft.com
URI: http://microsoft.com/