Centaur: A Hybrid Approach for Reliable Policy-Based Routing

Xin Zhang
Carnegie Mellon University
xzhang1@cs.cmu.edu

Adrian Perrig
Carnegie Mellon University/CyLab
perrig@cmu.edu

Hui Zhang
Carnegie Mellon University
hzhang@cs.cmu.edu

Abstract

In this paper, we consider the design of a policy-based routing system and the role that link state might play. Looking at the problem from a link-state perspective, we propose Centaur, a hybrid *routing protocol combining the benefits of both link state and path vector. Through analytical and experimental studies, we demonstrate Centaur's potential in achieving rich policy expressiveness and high network availability. Our work shows that it is possible to combine link-state and path-vector approaches into a practical and efficient algorithm for policy-based routing.*

1. Introduction

Path-vector routing is known for its slow convergence [12]. Link-state routing might seem comparatively advantageous by virtue of its faster convergence. However, link-state routing has been eschewed in a *policy*-based routing protocol design (such as inter-domain routing), due to its inability to support routing policies [5]. Consequently, most new inter-domain routing protocols have to implement policies through path-vector approaches, unavoidably bearing the limitation of inherent slow convergence.

In this paper, we revisit the design of a policy-based routing protocol and the role that link state might play, and show that it is possible to preserve the advantages of link-state routing while maintaining policy expressiveness. We start from a link-state protocol framework to inherit its fast convergence, and modify critical components of a traditional link-state protocol by using concepts from path vector to support routing policies. Using this framework, we design Centaur, a hybrid protocol between link state and path vector for reliable policy-based routing, as sketched below:

i) Network data model. Each node maintains a node-specific topology view, with each link in the topology annotated by routing policy predicates. A node N's topology view encodes *only* policy-compliant paths from N to other nodes in the network. This ensures that Centaur respects routing policies while preserving topology privacy.

ii) Link state announcement. Akin to LVA [4], Centaur employs link-level announcements as opposed to path-level announcements used in path vector. Differing from complete link-state flooding in traditional link-state protocols, in Centaur a node N passes onto a neighbor the link-state updates of only those links residing on policy-compliant paths. Such a selective link announcement enables rich routing policies, privacy, and scalability.

iii) Local solver. Instead of Dijkstra, a simpler algorithm is used to derive the policy-compliant paths.

Differing from HLP [18], LVA [4] and BGP-RCN [15] which also intend to enhance routing convergence by integrating link-state elements, Centaur notably enables routing policies via communicating only *enriched link-state* updates.

As an initial step, the current Centaur focuses on exchanging routing information between different Autonomous Systems (ASes), and aims to support basic routing polices, i.e., route filtering and ranking, under standard "customer/ provider/ peering" business relationships. For simplicity, we logically define a "node" in a Centaur topology as a sub-domain where the border routers have consistent exterior routing policies.

Through analytical and experimental studies, we demonstrate Centaur's potential in supporting rich routing policies while exhibiting better convergence and scalability than BGP. We hope this work will serve as a building block for future policy-based routing system design, and expand the design space of next generation routing protocols.

2. Challenges

In this section, we first highlight the fundamental challenges in adding policies to traditional link-state protocols (§2.1). Then we derive a key observation to motivate Centaur design (§2.2).

2.1. Basic Cases

Different Topologies. The correctness of link-state protocols depends on all nodes in the network having the same

This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and MURI W 911 NF 0710287 from the Army Research Office, and by NSF 100×100 project under grant ANI-0331653. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF, or the U.S. Government or any of its agencies.

view of the topology. Any discrepancy in the view of the network can cause routing loops in link-state protocols, as Figure 1 illustrates. Since path filtering policies essentially result in topology hiding, link-state protocols cannot implement filter policies in their current forms.

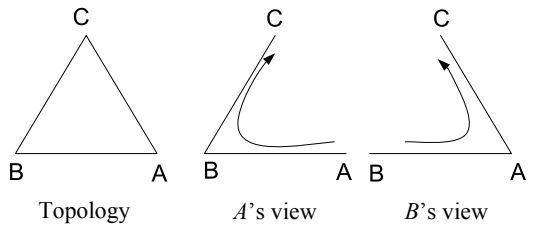
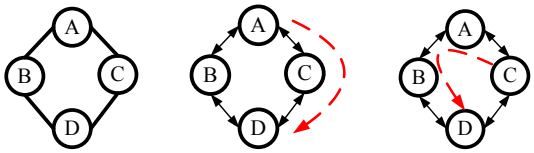


Figure 1. Loop in nodes with different topologies. Either A's or B's topological view contains only one path to C. To send a packet to C, A has to use B as the next hop, while B would send the packet back to A since B is not aware of the other path to C. This example shows that even on a simple topology, it is easy to form routing loops when nodes have different topologies.



(a) Topology. (b) A's view. (c) C's view.

Figure 2. Example topologies.

Diverse Policies. In traditional link-state protocols, all nodes in the network must use an identical algorithm to compute the paths, e.g., Dijkstra to compute the shortest paths. Therefore, the path ranking function has to be the same for all nodes, implying the lack of independence in path ranking policies; otherwise loops can arise.

To illustrate, consider the example in Figure 2 where node D is the only destination of interest. Suppose that due to local policies, C intends not to use its link $C \leftrightarrow D$ (the notation $\leftrightarrow$ indicates a bidirectional link) to reach D and does not announce it to node A. However in traditional link-state routing, node D announces this link to B and B floods it everywhere, whereby eventually A learns the link $C \leftrightarrow D$ from B, although C intends to hide it. If A and C choose different paths depicted as the dashed arrows in Figures 2(b) and 2(c) to reach D (conforming to their local policies), a loop will arise between A and C when either A or C attempts to send packets to D.

2.2. Analysis and Key Observation

We first define the terms *downstream* and *upstream* as follows.

Definition 1: In a routing path containing nodes N_1 and N_2 to a destination, if N_1 is closer to the destination than N_2, N_1 is downstream *of N_2 and N_2 is* upstream *of N_1 in the given path.*

With policy-based path hiding and ranking, given a link-state topology an upstream node may derive some paths that *do exist* in the topology but actually violate policies of the downstream nodes on the path (not being *used* by downstream nodes). This can potentially result in routing loops. For example in Figure 2, path $\langle A, C, D\rangle$ exists in A's topology view though it violates C's policy. However, node A does not know its downstream node C's routing path for reaching D, and hence cannot perform *loop detection* (an upstream node A will discard using a downstream path that already contains A).

This leads to an unsurprising yet motivating observation:

Observation 1: Routing loops can be effectively avoided if the upstream node U knows the downstream path so that U can perform loop detection.

Therefore, the problem of policy expression involves augmenting the link-state topology knowledge in each node with enough information to reflect the fact that only a subset of the paths in the link-state topology are ***policy-compliant*** routing paths (that are actually used by downstream nodes).

3. Centaur Overview

In this paper, we present Centaur in the context of traditional *single-path routing* for simplicity. §3.1 first summarizes Centaur features inherited from both link-state and path-vector paradigms. Then §3.2 sketches the high-level protocol flow and basic components of Centaur.

3.1. A Hybrid of Link State and Path Vector

Properties Derived from Link State. Akin to LVA [4], Centaur resembles link state in both topological representation of the network data model and link-level announcements (as opposed to path-vector announcements). In link-state protocols, nodes externally announce *individual* links, and internally maintain a consistent topology map from which routing paths are derived. Centaur retains this link-level announcement and topological representation, to achieve *faster convergence* and *lower update overhead* compared to path vector, as explained below.

In Centaur, when a route fails, nodes withdraw the specific link that caused the failure in the update message, rather than the entire route. The update message thus provides the exact fault location. Based on this ***root cause information***, other nodes can avoid exploiting alternative paths in their Routing Information Bases (RIBs) that also contain this failed link [6], [15]. By announcing individual links, Centaur can effectively eliminate a large amount of redundant information among the routing messages. For example, a

link that is on multiple paths needs only to be announced once.

Properties Derived from Path Vector. Centaur resembles path vector in both the way it implements policies, and the way it avoids loops. To enable policies, Centaur employs both link filters and local path rankings. Consequently, the topology views are different across different nodes. Centaur prevents loops despite the above topology and policy diversity, because each node N announces to an upstream node U only the best path that N uses by itself to reach a certain destination (policy-compliant path). In this way, U knows the path used by the downstream node, which satisfies Observation 1.

3.2. Protocol Overview

To better illustrate the basic concepts, we will go through a simple example in Figure 3 which enumerates the locally maintained routing states and routing-related operations for all the nodes in Figure 2(a)'s topology. For simplicity, we only consider the routing updates flowing from bottom up in the figure.

3.2.1. Link State Announcement.

Per Observation 1, we require that only the policy-compliant paths which are actually used by downstream nodes can be derived from the link-state topology. Therefore, we impose the first restriction that a node N announces only the links in the paths that N *uses* itself. We define such links as ***downstream links***. Downstream links are *directed*, with the direction from upstream pointing to downstream. Also, similar to BGP, destination nodes are explicitly marked in the announcements, associated with the prefixes owned by the destination nodes (which correspond to ASes in the inter-domain routing).

For example in Figure 3, node D announces downstream links $D \rightarrow B$ and $D \rightarrow C$ used to reach destinations B and C (where $\rightarrow$ denotes a directed link). Node B further announces to A only the links contained in its locally selected paths as presented in Figure 3.

Suppose C intends to hide its local link $C \rightarrow D$ from A to reach D. Note that B cannot announce link $C \rightarrow D$, because B only learned a *directed* link $D \rightarrow C$ from D (different from $C \rightarrow D$). Consequently, node A cannot derive a ***policy-violating path*** (that is not used by downstream nodes) $\langle A, C, D \rangle$ as it can in Figure 2(b).

We define the above way of only propagating directed downstream links as ***downstream link announcement***, which contrasts to the link flooding approach in traditional link-state protocols. As demonstrated above, downstream link announcement enables link filtering and path ranking policies while preventing loops (we discuss more complex cases in §3.2.4).

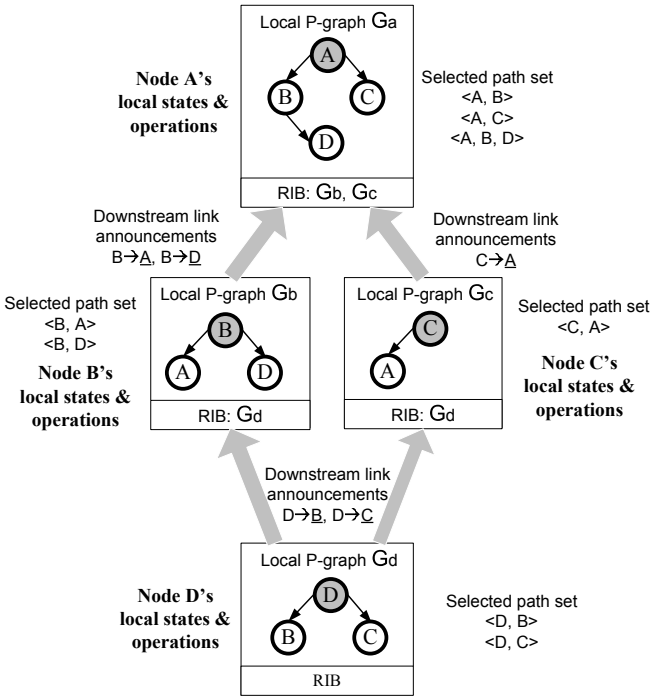


Figure 3. A simple example of Centaur operations and states. A gray node denotes the root of a P-graph. Nodes with underscores in the downstream link announcements are marked as destinations.

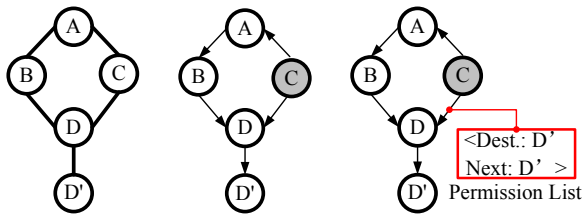
3.2.2. Network Data Model.

On receiving the downstream links from a neighbor N, the local node L stores them in its RIB as a directed graph rooted at N. We call such graph a ***P-graph (policy graph)***. In Centaur, a node stores in its RIB a P-graph per neighbor built from the downstream links exported by each neighbor. After the local node L selects its own path for each destination (as we show in §3.2.3), L constructs its local P-graph from the selected path set and stores its local P-graph. Figure 3 shows the P-graphs for all the nodes in Figure 2(a)'s topology. The P-graph data structure facilitates reassembling the downstream paths from the received downstream links, as presented below. §4.2 details the graph construction algorithm.

3.2.3. Local Solver.

Based on the downstream links learned from neighbors (as well as its own adjacent links), a local node L selects its own paths according to local ranking policies, and further announces the resulting downstream links. A straightforward path selection approach would be to first find all possible paths derivable from the local topology view and then choose the most preferred ones. Relevant algorithms are specified in §4.2 with the complexity analysis in §6.3.

Since the links exported by a neighbor (say N) are from



(a) Topology. (b) Invalid. (c) Permission List.

Figure 4. Permission List example. A gray node denotes the root of a P-graph.

the downstream paths that N uses, and N selects only one path for each destination (we assume single-path routing), the local node L can reconstruct exactly the same path set used by N from N's downstream link announcements, which satisfies Observation 1 and enables loop detection.

3.2.4. Permission List.

Downstream link announcements (§3.2.1) ensure that a P-graph contains only the links in policy-compliant downstream paths. However, in some scenarios policy-violating paths may still be derivable from the P-graph.

To illustrate, we extend the example in Figure 3 by adding a destination $D^{'}$ as Figure 4(a) depicts. Suppose C prefers $\langle C, A, B, D\rangle$ to reach D (rather than $\langle C, D\rangle$), but intends to use $\langle C, D, D^{'}\rangle$ to reach $D^{'}$ in which case $C \rightarrow D$ becomes a downstream link and is announced to A. The P-graph of C is now as Figure 4(b) depicts, and a policy-violating path $\langle C, D\rangle$ can be derived from the P-graph based on which A can further construct a policy-violating path $\langle A, C, D\rangle$.

In general, the presence of a ***multi-homed*** node (node with more than one parent, e.g., D in Figure 4(b)) in the P-graph will cause all its children including itself to have multiple derivable paths (both policy-compliant and -violating) from the root of the P-graph. For example, learning the P-graph in Figure 4(b) (more precisely, the corresponding link set) from neighbor C, the upstream node A cannot tell which one of the paths is actually used by C to reach D, thus breaching Observation 1. Our approach is to embed the restrictions via ***Permission Lists*** onto appropriate links to further eliminate policy-violating paths. A permission list on a link is basically a set of paths that can use this link. §6.1 further proves the policy expressiveness with Permission Lists.

An example is shown in Figure 4(c). The Permission List specifies that only the path in which (i) the "Destination is $D^{'}$" and (ii) the "Next Hop of the *multi-homed* node (i.e., D) is $D^{'}$" is policy-compliant.

4. Centaur Details

In this section we first elaborate on Permission List (§4.1), the key data structure in Centaur. We then present the operational algorithms on the P-graphs (§4.2), and finally describe the protocol flow of Centaur (§4.3).

4.1. Permission List Specification

A Permission List is attached to a link $A \rightarrow B$ when B becomes multi-homed in a certain P-graph, set by the *creator* of the P-graph during its local P-graph construction. For example in Figure 4(c), C sets a Permission List to link $C \rightarrow D$ since node D becomes multi-homed (has more than one parent) in C's local P-graph. The Permission List of a link $A \rightarrow B$ represents the set of *all and only* derivable *policy-compliant* paths, denoted by $\mathcal{P}_{A\rightarrow B}$, that pass through $A \rightarrow B$ in a P-graph.

To represent the path set $\mathcal{P}_{A\rightarrow B}$, a naive way is to create one entry in the Permission List for every policy-compliant path that traverses the link $A \rightarrow B$. This *exhaustive per-path encoding* is theoretically useful in demonstrating the expressiveness of Permission Lists (§6.1). In practice, Permission List entries can be simplified by using a *per-dest-next encoding*, where each policy-compliant path can be represented by a $\langle Destination, NextHop\rangle$ pair. More specifically, given a link $l = A \rightarrow B$ (B is the multi-homed node), each policy-compliant path p containing link l can be uniquely identified by the destination of path p, and the next hop of B in path p. It is not difficult to prove that per-dest-next encoding has the same descriptiveness as exhaustive per-path encoding.[1]

Hence, $\mathcal{P}_{A\rightarrow B}$ can be a list of $\langle Destination, NextHop\rangle$ pairs rather than full path descriptions. In a Permission List, destinations with the same next hop can be grouped into one pair entry where the $Destination$ field is actually a list of destinations. Clearly, the size of a Permission List (the number of destinations in it) is bounded by the number of downstream destinations in a given P-graph. Fortunately, evaluation results in §5 indicate that the Permission List size can be much smaller in realistic scenarios. In addition, we can also adopt simple compression schemes (such as Bloom Filters) to compactly represent a list of destinations.

4.2. P-graph Operations

In this subsection we specify the two major operations on a P-graph: deriving paths from a P-graph assembled from a neighbor's downstream link announcements, and constructing a local P-graph with Permission Lists from a locally selected path set. In §6.3 we analyze the time complexity of the algorithms.

Derive paths from P-graph. In Centaur, we need to derive policy-compliant paths instead of shortest paths. Given the *directed* links and Permission Lists in a P-graph, it ensures that there is *only one* policy-compliant path for each destination derivable from the P-graph (in the single-path routing

1. Due to space limitation, we defer the proof to the full version.

Table 1. $DerivePath$ algorithm to derive the path for a given destination D in a P-graph

line	action
1	Set *currentNode* = D
	//Backtrace to the root
2	**while** *currentNode* != *root*
3	**if** *currentNode*.multiHomed == $false$
4	*traversedRoute*.add(*currentNode*)
5	*currentNode* = *currentNode*.getParent()
6	**else**
7	**for** each link l pointing to *currentNode*
8	**if** l.permissionList.Permit(D,*currentNode*)
9	*currentNode* = *currentNode*.getParent(l)
10	*traversedRoute*.add(*currentNode*)
11	**break**
12	**return** *traversedRoute*

case). To reconstruct the path for a certain destination, the $DerivePath$ algorithm simply starts from the destination node and follows the parent chain in the P-graph under Permission List restrictions, until the root of the P-graph is reached. Table 1 describes the $DerivePath$ algorithm.

Construct Local P-graph with Permission Lists. For each destination D, the local node first derives all the paths to D from the RIB as well as its local adjacent links (if any), and then applies local preferences to select the best path for D, and finally adds the links in the path to the local P-graph. If a multi-homed node B appears after adding a link $l = A \rightarrow B$ in path p for reaching destination D, the algorithm updates (or creates) the Permission List of l with an entry set to $\langle DestList.Add(D), Next : N\rangle$ where N is the next hop of B in path p. Table 2 summarizes such $BuildGraph$ algorithm.

Table 2. $BuildGraph$ algorithm to construct a P-graph.

line	action
1	**for** each destination D
2	*pathSet* = getAllPath(RIB,D)
3	*path* = localPreference(*pathSet*)
4	**for** each link $l = A \rightarrow B$ in *path*
5	graph.Add(l)
6	**if** B.inDegree > 1
7	l.permissionList.Add($D, B.getNextHop($*path*$)$)

4.3. Centaur Protocol Flow

Notation. In Centaur, a node A maintains a local P-graph, denoted by G_A. Let $\mathbb{N}(A)$ stand for the set of A's neighbors. Node A also stores in its RIB the P-graph assembled from each neighbor's downstream links announcements, denoted by $G_{B\rightarrow A}$, $\forall B \in \mathbb{N}(A)$. A policy tuple $\langle Imp_A, Exp_A, Pref_A\rangle$ of node A contains the *import* and *export filters* operating on links, and *local preference* to rank candidate paths.

We formulate the protocol flow in two phases as follows.

4.3.1. Initialization Phase.

Step 1. During initialization, a node A first becomes aware of its adjacent links and constructs the corresponding downstream links pointing to B, $\forall B \in \mathbb{N}(A)$. Before advertising these links, A employs export filters to hide certain links from some neighbors according to A's local policies.

Step 2. On receiving downstream link announcements from a neighbor B, a node A first removes the links which point to A itself $(X \rightarrow A|X \in \mathbb{N}(A))$ for loop elimination, and also employs import filters (Imp_A) to remove undesired links according to A's local policies. Subsequently, A organizes and stores the links received from B as a P-graph in A's RIB, denoted by $G_{B\rightarrow \mathrm{A}}$. Using our notation, during initialization (the $\backslash$ operator represents the standard set difference operator):

$$G_{B\rightarrow \mathrm{A}} = Imp_A\big(Exp_B(G_B)\backslash\{X \rightarrow A|X \in \mathbb{N}(A)\}\big)$$

Step 3. A node then evokes a $BuildGraph$ procedure (Table 2) based on the P-graphs in its RIB as well as its local adjacent links to create its local P-graph.

Step 4. After constructing the local P-graph, a node A keeps on announcing the links from the P-graph pruned by export filters. Differing from Step 1, in this step node A announces not only its adjacent links, but also the downstream links learned from A's neighbors; and if needed, a link is associated with a Permission List. In both Steps 1 and 4, destination nodes (prefixes in practice) are marked in the link announcements.

4.3.2. Steady Phase.

Step 5. After the initialization phase, a node N in the network sends update messages incrementally on a per-*link* basis when a change of a certain link state occurs due to either link failures or *policy changes* (say, a link is no longer in any of N's preferred paths).

Let Δ_B denote the links added or *removed* in node B's locally selected path set. In order to calculate Δ_B, when constructing its local P-graph, node B needs to associate a counter with every link in the P-graph, recording how many selected paths contain each given link. When the counter value of a certain link decreases to zero (indicating no selected path contains this link any longer), the link is included in Δ_B as to be removed.

Just like in the initialization phase, both the sender B of the update and the receiver A of the update messages apply their respective export and import filters to Δ_B. Let $G_i^{'}$ be the updated version of P-graph G_i. Node R then merges the received changes into $G_{B\rightarrow A}$ which already exists in A's RIB. Let $G^{'}_{B\rightarrow A}$ stand for A's updated view of $G_{B\rightarrow A}$. $G^{'}_{B\rightarrow A}$ is calculated as follows:

$$G^{'}_{B\rightarrow A} = Imp_A\big(Exp_B(\Delta_B)\backslash\{C \rightarrow A|C \in \mathbb{N}(A)\}\big) \cup G_{B\rightarrow A}$$

During updating the P-graph, if a previously multi-homed node turns into single-homed, a corresponding Permission List is removed. Similarly, a Permission List will be created if a multi-homed node appears.

5. Evaluation and Implementation

In this section, we evaluate the scalability of Centaur by experimenting with measured AS topologies, and implement a prototype on the Distcomm platform [1] to show Centaur's rapid convergence and overhead reduction from BGP.

5.1. Methodology

We consider two major metrics for the evaluation: (i) the *communication overhead* of routing update messages, and (ii) the *convergence time* after link failures. In the following we explain each key component in our evaluation in turn.

Input Topologies and Policies. Our simulation is conducted on three sources of topologies. First, we use multiple inter-AS topologies obtained from both CAIDA [7] and He et. al [8] ("HeTop"), annotated with business relationships on the links. Both CAIDA and HeTop take RouteViews [9] snapshots as input, and infer business relationships between nodes. HeTop augments its inference with various other data sources, and finds more peering links. Due to space limitation, we present only one topology from each of CAIDA and HeTop; experiments on other topologies yield similar observations. We summarize the characteristics of selected topologies in Table 3. We also use BRITE [13] to generate topologies for running a prototype of Centaur on the DistComm platform [1].

Throughout our experiments, all the update message propagation and path derivation will follow the business policies annotated in the input topologies. Since it is an open problem to accurately infer the *intra*-AS structure and policy, the best we can do is to model each AS as a node in the network.

Table 3. Characteristics of input topologies.

Name/Date	Node/Link	Peering/Provider/Sibling
CAIDA/ Sep'07	26022/ 52691	4002/ 48457/ 232
HeTop/ May'05	19940/ 59508	20983/ 38265/ 260

Communication Overhead. We consider the *message count* metric, which represents the number of messages that need to be generated because of an event such as a link going down. We also present the sizes and population the Permission Lists.

Convergence Time. To study the dynamic convergence behavior of Centaur, we implement a prototype of Centaur to make direct comparison with standard path-vector and link-state protocols (BGP and OSPF in our experiment). We study the *convergence time* required for the network to get re-stabilized (i.e. no further update messages are sent) after we randomly remove or add a link in the network.

Table 4. Structural characteristics of P-graphs.

	CAIDA	HeTop
No. of links	40339	32006
No. of Permission Lists	14437	12219

Table 5. # entries of Permission Lists.

	# entries=1	# entries=2	# entries=3	# entries>3
CAIDA	0.7%	91.9%	7%	0.6%
HeTop	0.7%	92.9%	6.4%	0.1%

5.2. Measurements with AS Topologies

For each node in a given AS topology, we first derive a complete path set reaching all other nodes in the topology, according to the standard business relationship. Then we build the local P-graph for each node from its path set. Table 4 presents the average number of links with Permission Lists, and Table 5 presents the distribution of the number of entries (pairs of destination list and next hop) in one Permission List. Note that we do not count the number of total destinations contained in Permission Lists, since we assume all the destinations in one destination list of a single entry can be compactly represented using Bloom Filters.

In Figure 5 we measure the number of update messages triggered as an immediate result of a single link failure (averaged over all possible link failures for each AS). In this example we do not consider the cascading effects of propagating updates. The results indicate Centaur incurs roughly 100 to 1000 times fewer update messages.

5.3. Prototype and Results

We implement Centaur on the DistComm platform. DistComm is a session-level BGP simulator built on the SSFNet

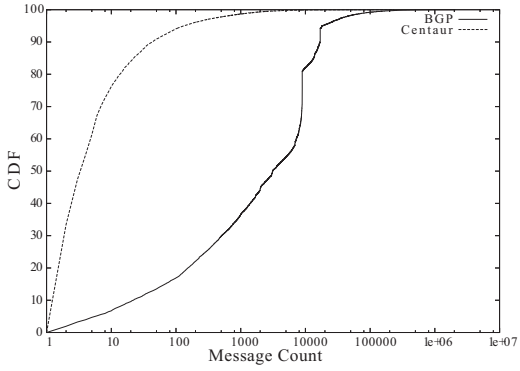


Figure 5. Immediate overhead of single link failure, RV topology

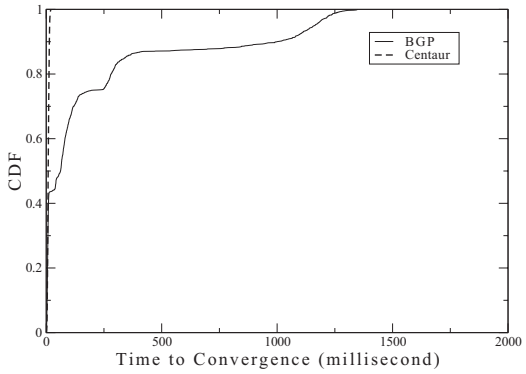


Figure 6. Convergence time comparison.

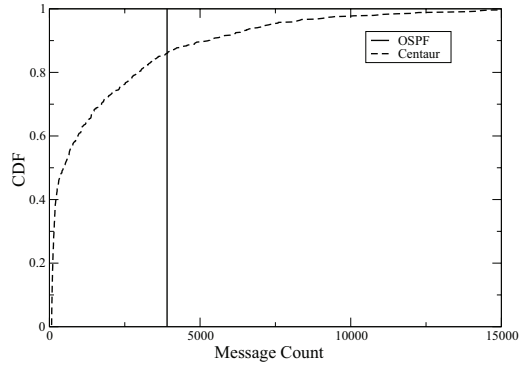


Figure 7. Convergence load comparison.

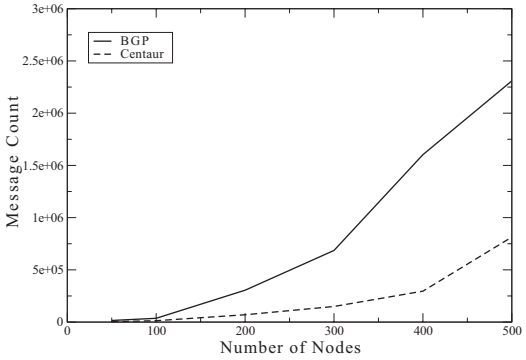


Figure 8. Scalability.

code base [2]. Unfortunately our current computing resource cannot run the prototype on a large number of nodes simultaneously, therefore we use the BRITE topology generator to create a network topology with 500 nodes. Though imperfect, the results from such a small topology still present interesting convergence and update behaviors of Centaur, as well as Centaur's potential in much larger topologies.

Since Centaur is a policy-based routing protocol, we infer the standard "customer-provider" business relationships between nodes using the location of the nodes in the topology. We set the nodes at the center of the topologies (the nodes with largest degrees) to be Tier-1 provider, the nodes below them to be Tier-2 and so forth.

To measure the convergence behavior of Centaur, we let a 500 node topology stabilize and then we sequentially "flip" each link in the topology, i.e., first remove the link and wait till the routing protocol converges; then bring the link back up and wait for the convergence again. After each flip we measure the total count of messages sent and the duration time required to re-stabilize. We ignore the CPU delay while the link delays are generated automatically in the BRITE topology file. They are set randomly between 0 and 5 milliseconds.

Figure 6 shows the CDF of convergence time of both protocols. Clearly Centaur converges much faster than BGP almost all the time. In Figure 7, we compare the network load of flipping links against OSPF. We observe that Centaur converges with fewer message count than OSPF for 82% of the cases. This can be explained by noting that OSPF does not implement policies, so every link's information needs to be transmitted over every other link in the network. To investigate Centaur's potential in larger topologies, we create topologies of various sizes and cold start the protocols until they stabilize. In Figure 8, we give the update overhead of Centaur and BGP under different topology sizes given a routing update event. It is apparent that Centaur presents more distinct advantage on larger topologies.

6. Properties and Limitations

In §5, we experimentally evaluate Centaur's convergence rate and communication overhead. In this section we further analytically investigate Centaur's policy expressiveness, privacy, and computational complexity.

6.1. Policy Expressiveness

In path vector, an important means of implementing routing policies is through ***selective path announcement***. For example, by announcing only a selected *subset* of its available paths, a node N can realize filtering policy and path ranking policy (hiding the undesirable paths). We claim that:

Claim 1: Along with the basic import/export filters and local preferences, the Permission Lists are capable of capturing the full policy expressiveness of selective path announcement in path vector.

Proof: We start with proving the policy expressiveness of exhaustive per-path encoding (§4.1). Consider that a node N announces a path p to an upstream node U. This routing behavior can be described by the Permission List entry of link $U \rightarrow M$: p. This proves that by using exhaustive per-path policy encoding, selective Permission Lists can achieve all policies carried by any selective path-vector set. Since per-path and per-dest-next Permission Lists are equivalent, the claim follows. □

Limitation. Note that there are other classes of policies that can be implemented beyond selective path announcement. For example in BGP, peering agreements and community attributes can be employed to implement more flexible and complicated policies. In the future work, we intend to consider how Centaur can implement these policies, and how Centaur can cope with iBGP-like intra-domain infrastructure to implement *finer*-grained policies.

6.2. Privacy

It may appear that Centaur forces nodes to directly reveal both topology connectivity (via P-graphs) and routing policies (via Permission Lists). We show that in fact this does not happen. We make the following observations:

Claim 2: Due to the routing constraints carried in the Permission Lists, Centaur reveals the same topological information to BGP. If a routing policy is disclosed by the Permission Lists in Centaur, it can also be uncovered in BGP with additional complexity. Therefore Centaur does not expose more topological privacy nor policy privacy compared to path-vector protocols.

Proof: For the ease of explanation, we consider exhaustive per-path encoding. Each entry in the Permission Lists exactly corresponds to a path vector, thus establishing a one-to-one mapping between the derivable path set in Centaur to the selected path set in path vector protocols. Since both Centaur and path vector protocols contain the same path set, the derivable topological information is identical.

Suppose the routing information in Centaur is now propagated in the form of path vectors. Then we can construct exactly the same P-graph and Permission Lists as those in Centaur from the given path vector set using the $BuildGraph$ procedure (Table 2). This proves that we can learn the same amount of policies through the constructed Permission Lists, with the additional complexity of constructing a P-graph from the given path vector set. □

A positive and important note is that, *the Permission Lists do not necessarily leak a specific node's policy*. For example in Figure 4(c), the Permission List on link $C \rightarrow D$ forbids the traffic to destination D via that link. However, it might be the policy of several possible nodes, such as A or C.

From the claim above we can also derive an interesting insight as follows: *From the perspective of routing information expressed, Centaur is equivalent to a path vector protocol that includes root cause notification and in which the format of the information passed between nodes is compressed.*

6.3. Computational Complexity

In Centaur, the set of paths are implicitly encoded in a graph topology, thus it introduces additional computational overhead in constructing the graph and deriving paths from the graph. Suppose a P-graph has $|E|$ links. The time complexity of $BuildGraph$ procedure (Table 2) is given by $O(|E|\alpha)$, where α is the time complexity of inserting a single link into a graph, which depends on the implementation of the graph data structure (e.g., adjacent links or matrix). The time complexity of $DerivePath$ algorithm (Table 1) is given by $O(d \times i)$, where d is the length of the path and i is the in-degree of a node in the path.

Also note that, in Centaur each node needs to maintain the P-graph data structure and the space complexity depends on different implementations of the graph data structure. As we pointed in §4.1, the Permission List can be efficiently compressed. Also, the result in Table 5 indicates that the size of a Permission List in practice is small.

6.4. (De)Aggregation and Isolation

Centaur mainly addresses the *dissemination* of routing updates, which is orthogonal to the *granularity* of the routing updates. In Centaur, a node can announce its owned prefixes at any aggregate or de-aggregate level in the same way as BGP. For example in Fig 2(a), node D (presumably corresponding to an AS) can announce separate fine-grained prefixes for D's multiple sub-nets (in which case node D can be logically split into multiple "node"s in the topology views in Centaur), or one single aggregate prefix representing the whole domain. By choosing appropriate prefix aggregation levels, Centaur can achieve routing update isolation in the same way as BGP does.

7. Related Work

In this section we first examine related efforts on link-state routing, then summarize other lines of endeavors for designing a new routing protocol.

Link-State Attempts. In LVA [4], the authors use *link vectors* to achieve better scalability than link-state protocols. However unlike Centaur, LVA does not implement rich policies. White et al. suggest using the combination of path vector, topological graph overlay, and SoBGP [19]. The graph overlay is however applied on top of standard path-vector BGP as a layer of additional information, whereas in Centaur the graph topology is central to the routing algorithm. BGP-RCN also provides link-level failure information. However, it is built on top of BGP, the additional link-level information must be piggybacked onto path-level update messages.

In HLP [18], the authors propose to divide the Internet into two levels: a top level running BGP and a bottom level running a link-state protocol. They provide isolation and scalability using the two-level hierarchy, and provide fast convergence by relying on the link-state portion. However, HLP requires exceptions for complex policies and cannot express policies in the link-state level. Furthermore, HLP can be orthogonal to our work in the sense that we can

envision to replace BGP by Centaur at the top level in HLP where routing policies exist.

Multi-Path Routing. Researchers also explored multi-path routing for enhancing network availability [10], [11], [14], [17], [20], [22]. While most previous multi-path routing efforts are built upon path vector and orthogonal to our study, we anticipate Centaur may better support multi-path routing since it can propagate multiple paths for a destination in a more compact and scalable way.

Source Routing. Other researchers also advocated source-route based approaches [3], [16], [21], [23], where the edge routers or even end users can specify the relaying nodes along the forwarding path. Using such source routing relieves the core routers of computing the paths for destinations, however it hampers the ISPs' ability to control the traffic.

8. Conclusion and Future Work

In this paper we demonstrate the feasibility of adopting a new link-state perspective for designing a policy-based routing protocol. Using a standard link-state protocol model as the starting point, we derive Centaur, a hybrid protocol which supports rich policies by integrating path-vector functionality. We show analytical and experimental results indicating that Centaur is a promising approach to enhancing network availability with good scalability. We hope this work can open new research directions on designing policy-based routing incorporating a link-state perspective.

9. Acknowledgements

The authors gratefully thank Debabrata Dash and Haowen Chan for their collaboration at various stages of the project, and the anonymous reviewers for their valuable feedback.

References

[1] distcomm network simulation environment. http://www-users.cs.umn.edu/~jkrasky/distComm.

[2] Scalable simulation framework. http://www.ssfnet.org/.

[3] K. Argyraki and D. R. Cheriton. Loose source routing as a mechanism for traffic policies. In *ACM SIGCOMM Future Directions in Network Architecture*, 2004.

[4] J. Behrens and J. J. Garcia-Luna-Aceves. Distributed, scalable routing based on link-state vectors. In *ACM SIGCOMM*, 1994.

[5] M. Caesar and J. Rexford. BGP routing policies in ISP networks. *IEEE Network Magazine*, Special issue on interdomain Routing, Dec 2005.

[6] J. Chandrashekar, Z. Duan, Z.-L. Zhang, and J. Krasky. Limiting path exploration in BGP. In *IEEE INFOCOM*, 2005.

[7] X. Dmitiropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. Claffy, and G. Riley. AS relationships: Inference and validation. *ACM CCR*, 2007.

[8] Y. He, G. Siganos, M. Faloutsos, and S. V. Krishnamurthy. A systematic framework for unearthing the missing links: Measurements and impact. In *USENIX NSDI*, 2007.

[9] http://www.routeviews.org/. University of oregon route views project.

[10] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. Bananas: An evolutionary framework for explicit and multipath routing in the internet. In *ACM SIGCOMM Future Directions in Network Architecture*, 2003.

[11] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs. R-BGP: Staying Connected In a Connected World. In *USENIX NSDI*, 2007.

[12] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. *IEEE/ACM Trans. Netw.*, 9(3):293–306, 2001.

[13] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *MASCOTS*, 2001.

[14] M. Motiwala, N. Feamster, and S. Vempala. Path splicing: Reliable connectivity with rapid recovery. In *ACM HotNets*, 2007.

[15] D. Pei, M. Azuma, D. Massey, and L. Zhang. BGP-RCN: improving bgp convergence through root cause notification. In *Technical Report, UCLA CSD TR-030047*, 2003.

[16] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *ACM SIGCOMM*, 2004.

[17] I. Stoica and H. Zhang. LIRA: An approach for service differentiation in the Internet. In *Nossdav*, 1998.

[18] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: a next generation inter-domain routing protocol. In *ACM SIGCOMM*, 2005.

[19] R. White. Graph overlays on path vector. *Internet Protocol Journal, Cisco*, 8(2):13–21, June 2005.

[20] W. Xu and J. Rexford. MIRO: Multi-path Interdomain Routing. In *ACM SIGCOMM*, 2006.

[21] X. Yang. NIRA: A new routing architecture. In *ACM SIGCOMM FDNA Workshop*, 2003.

[22] X. Yang and D. Wetherall. Source Selectable Path Diversity via Routing Deflections. In *ACM SIGCOMM*, 2006.

[23] D. Zhu, M. Gritter, and D. Cheriton. Feedback based routing. In *ACM Hotnets Workshop*, 2002.