University of Pennsylvania
ScholarlyCommons

Departmental Papers (ESE) | Department of Electrical & Systems Engineering

June 2008

Message-Efficient Dissemination for Loop-Free Centralized Routing

Haldane Peterson
University of Minnesota

Soumya Sen
University of Pennsylvania, soumyas@princeton.edu

Jaideep Chandrashekar
Intel Corp.

Lixin Gao
University of Massachusetts

Roch A. Guérin
University of Pennsylvania, guerin@acm.org

See next page for additional authors

Follow this and additional works at: http://repository.upenn.edu/ese_papers

Recommended Citation

Haldane Peterson, Soumya Sen, Jaideep Chandrashekar, Lixin Gao, Roch A. Guérin, and Zhi-Li Zhang, "Message-Efficient Dissemination for Loop-Free Centralized Routing", . June 2008.

Reprinted from *ACM Computer Communication Review*, Volume 38, Issue 3, July 2008, pp. 65-74.
Publisher URL: http://doi.acm.org/10.1145/1384609.1384616

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/ese_papers/420
For more information, please contact repository@pobox.upenn.edu.

Message-Efficient Dissemination for Loop-Free Centralized Routing

Abstract
With steady improvement in the reliability and performance of communication devices, routing instabilities now contribute to many of the remaining service degradations and interruptions in modern networks. This has led to a renewed interest in centralized routing systems that, compared to distributed routing, can provide greater control over routing decisions and better visibility of the results. One benefit of centralized control is the opportunity to readily eliminate transient routing loops, which arise frequently after network changes because of inconsistent routing states across devices. Translating this conceptual simplicity into a solution with tolerable message complexity is non-trivial. Addressing this issue is the focus of this paper. We identify when and why avoiding transient loops might require a significant number of messages in a centralized routing system, and demonstrate that this is the case under many common failure scenarios. We also establish that minimizing the number of required messages is NP-hard, and propose a greedy heuristic that we show to perform well under many conditions. The paper's results can facilitate the deployment and evaluation of centralized architectures by leveraging their strengths without incurring unacceptable overhead.

Keywords
Network, Routing, Loops, Graph, Complexity

Comments
Reprinted from *ACM Computer Communication Review*, Volume 38, Issue 3, July 2008, pp. 65-74.
Publisher URL: http://doi.acm.org/10.1145/1384609.1384616

Author(s)
Haldane Peterson, Soumya Sen, Jaideep Chandrashekar, Lixin Gao, Roch A. Guérin, and Zhi-Li Zhang

This journal article is available at ScholarlyCommons: http://repository.upenn.edu/ese_papers/420

Message-Efficient Dissemination for Loop-Free Centralized Routing*

Haldane Peterson
University of Minnesota
peterson@cs.umn.edu

Soumya Sen
University of Pennsylvania
ssoumya@seas.upenn.edu

Jaideep Chandrashekar
Intel Research
jaideep.chandrashekar@intel.com

Lixin Gao
University of Massachusetts
lgao@ecs.umass.edu

Roch Guerin
University of Pennsylvania
guerin@ee.upenn.edu

Zhi-Li Zhang
University of Minnesota
zhzhang@cs.umn.edu

ABSTRACT

With steady improvement in the reliability and performance of communication devices, routing instabilities now contribute to many of the remaining service degradations and interruptions in modern networks. This has led to a renewed interest in centralized routing systems that, compared to distributed routing, can provide greater control over routing decisions and better visibility of the results. One benefit of centralized control is the opportunity to readily eliminate transient routing loops, which arise frequently after network changes because of inconsistent routing states across devices. Translating this conceptual simplicity into a solution with tolerable message complexity is non-trivial. Addressing this issue is the focus of this paper. We identify when and why avoiding transient loops might require a significant number of messages in a centralized routing system, and demonstrate that this is the case under many common failure scenarios. We also establish that minimizing the number of required messages is NP-hard, and propose a greedy heuristic that we show to perform well under many conditions. The paper's results can facilitate the deployment and evaluation of centralized architectures by leveraging their strengths without incurring unacceptable overhead.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers

General Terms

Measurement, Control, Performance, Reliability

Keywords

Network architecture, Network routing, Network management, Graph theory, Algorithms, Message complexity

1. INTRODUCTION

Motivated by the challenges of managing and controlling distributed routing decisions in IP networks, there has recently been a renewed interest in centralizing important parts of the routing function [7, 2, 14]. The goals of such centralization are twofold. First is to simplify routing infrastructure by refactoring its function to minimize the expense of distributed switches (*forwarding elements* or *FEs*). The second goal is to make the network more responsive, manageable, and flexible by placing complex decision functions in a central resource, rather than composing them from distributed routing protocols.

This paper uses the term *Centralized Routing Networks* or *CRNs* to refer to network architectures with centralized routing decisions. For example, the 4D architecture [7] replaces the path selection component of an IGP with a *Decision Plane* running on a Decision Element (*DE*)[1]. This architecture simplifies FEs by removing both the memory necessary to store global network state and the processing power to compute forwarding paths. It also enables more sophisticated path computations, because paths need not be calculated based on global coordination. All aspects of path selection are under the sole control of the DE.

However, a centralized architecture is not a routing panacea. There remains a gap between the inherently distributed nature of the forwarding network of FEs and the centralized decision element. In the 4D architecture, this gap is bridged by a *Dissemination Plane,* a component to facilitate communication between FEs and DE. Dissemination solves a problem analogous to that solved by flooding in distributed link-state IGPs: it distributes routing information about the network from where it is observed or created to where it is needed. But Dissemination introduces unavoidable communication delays between events in the network and decisions at the DE to accommodate them, and further delays between those decisions at the DE and their implementation on the FEs. These delays occur irrespective of how Dissemination is realized, *e.g.,* in-band or out-of-band.

This paper identifies several challenges that arise from this architectural gap, and explores solutions to them. We focus on design issues in the Dissemination plane and its interface with the Decision plane. We pay particular attention to the message complexity of Dissemination and its optimization, including development of message complexity metrics.

1.1 Problem: Transient Forwarding Loops

A network is dynamic: nodes and links change state over time, and those changes can force changes to previously se-

*Supported by NSF grants CNS-0626617, CNS-0627004, and CNS-0626808.

[1]For scalability, performance, and reliability it is likely that a DE would be implemented as a distributed service across multiple nodes spread through the network. The point of the term "centralized" is that it is not implemented on the FEs, which have a distinct set of architectural functions.

lected forwarding paths. Modifying existing paths in reaction to network changes introduces the potential for transient forwarding loops. Traffic is caught in a forwarding loop when, after detecting and reacting to a network change, FEs choose inconsistent next-hops that create cycles in source-destination paths. This is a long-standing concern in large operational networks, where loops occur frequently in practice [8]. There is a rich literature on reduction and elimination of forwarding loops, *e.g.,* [6, 3, 5], existing routing protocols that guarantee the avoidance of loops [1]; and ongoing standardization efforts [4] to augment existing protocols with such a capability. Among this vast literature, [3] is most relevant to the problem of avoiding transient loops in a CRN, because of their focus on link-state routing protocols that share a similar information and computational model with a CRN (albeit one that is replicated across routers rather than present only at the DE itself). As a result, some loop avoidance techniques derived for link-state protocols are readily applicable to a centralized routing setting. In particular, [3] has shown that transient loops can be eliminated by imposing a specific order on updates of the Forwarding Information Bases (FIBs[2]) of FEs in the network.

The current literature on loop avoidance has a dual focus: first, defining constraints on FIB update order that guarantee freedom from loops; and second, developing implementations to enforce those constraints with minimal modifications to existing distributed routing protocols. Loop-free FIB update ordering is a fundamental property of network forwarding, and applies just as well to CRNs as to networks routed with IGPs. However, implementation considerations change substantially in a CRN. Techniques to coordinate FIB updates with distributed IGPs rely on mechanisms that are unnecessary and onerous in a CRN, and the centralized operation of a DE presents opportunities to simplify and control the update process. The result suggests implementation approaches quite different from what has been described previously in the literature. [5] deals with the implications of a CRN but stops short of specifying loop-free update orders.

Specifically, loop-avoidance solutions aimed at IGPs involve two components: incremental adjustment of link metrics, and control messages to trigger updates. Both components reflect the need for coordination of FIB updates among FEs, and for minimizing modifications to existing distributed routing protocols. But such constraints do not apply to a DE, which operates on centralized computations, and which need not be concerned with protocol compatibility because CRN protocols will, of necessity, be new. Furthermore, given the stated CRN goal of minimizing the decision making ability of FEs, the DE can define and enforce the order in which FEs update their FIBs, and hence eliminate the need for extensive message exchanges among FEs.

Consider a simplified, centralized approach to transient loop avoidance. After a network change, let the DE compute the order in which FEs should update their FIB, and then use mechanisms designed into the Dissemination protocol to ensure that FEs perform these updates in the computed order. While such a solution is clearly possible, it may be highly inefficient, to the point of being operationally impractical. This is because, as we shall see, a naïve solution for determining an appropriate ordering of FIB updates can result in a very large number of update messages. In a CRN, the number of messages required to update FIBs after a change is of particular importance, as all of these messages originate from the DE, that then becomes a performance bottleneck. Both the potential for transient forwarding loops and the performance bottleneck of Dissemination messages hold whether the DE is a fully centralized or a distributed service. Devising a scheme for computing at the DE or DEs an ordering of FIB updates that can both avoid loops and minimize the number of messages needed is, therefore, important to provide this functionality efficiently in a CRN.[3]

[2]The FIB of an FE is a list of entries, where each entry specifies a destination set and the next-hop(s) to which packets that match the entry can be forwarded.

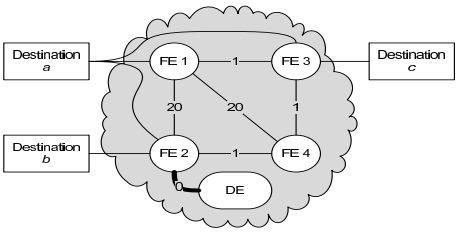

Figure 1: Basic network model: Forwarding Elements, Decision Element, Links, Destinations, and Egresses.

In this paper, we devise and evaluate such a scheme based on grouping of FIB updates to avoid loops while minimizing the number of individual messages required. Using simulations and network traces, we demonstrate the need for such a scheme by identifying when and why simple scenarios can give rise to conditions where multiple update messages per FE are required.[4] We evaluate the performance of our update avoidance scheme with both analysis and simulations across various topologies and failures scenarios.

The rest of this paper is structured as follows. Section 2 presents a model for a centralized routing system, with notations and assumptions used in the following analysis. Section 3 describes the problems of loop-free centralized routing, in particular with respect to message optimization. We identify a condition, which we call *No Ordering*, which precludes the loop-free update of all nodes in a single pass. Section 4 searches for instances of the No Ordering condition in real and simulated networks. Section 5 presents and evaluates a Greedy algorithm for consolidating updates to minimize message complexity. Section 6 presents conclusions based on all of the results, analyses, and observations of the paper.

2. FORMAL MODEL

We model a network as a directed graph $G = (V \cup \{DE\}, E)$, where V is the set of FEs in the network, and

[3]The potential convergence latency introduced by a centralized decision service is beyond the scope of this paper, but we do note some implications to our solution to the transient loop problem. Given that convergence delays are inherent to CRNs, and CRNs have considerable flexibility in path selection, it is reasonable to suppose that a CRN would have path protection mechanisms.

[4]For simplicity, this model assumes that an FE can update its FIB incrementally, that is, the FIB entry for any destination is writable independently of other destinations in the FIB. Each update message sent by the DE, then, need affect only a subset of destinations the FIB.

DE corresponds to the centralized route service. Edges $(v_i, v_j) \in E(G)$ represent unidirectional links. We assume that the network topology is reasonably *robust*, in the sense that a single or a few link failures will not partition the network[5]. In particular, to ensure that the DE is never disconnected from the network, we assume that the links connecting it to its adjacent FEs never fail simultaneously. While partitioned networks and DE failures are important problems in operational networks, they are beyond the scope of this paper.

The network has a set D of *destinations*, corresponding to IP prefixes[6]. Each reachable destination d is associated with one or more FEs, called *egress nodes* for d. In Figure 1, FEs 1, 2, and 3 are all egresses for destination a. FE 2 is the unique egress for destination b, and FE 3 is the unique egress for destination c. FE 4 is not an egress for any destination.

The goal of the routing system is to specify at least one path from each FE to each reachable destination. In a sophisticated network where it is possible (and desirable) to perform multipath routing (for load balancing, QoS, traffic engineering, etc.), the FIB of a given FE may contain multiple neighbor nodes as next-hops for a single destination. For destinations for which an FE is the egress node, *i.e.,* a directly attached subnet or a prefix reachable through an eBGP peer, the FIB entry points to the local interface on which to forward the packet and is not associated with a "neighbor" FE in V.

Thus in the most general scenario, the forwarding decisions implemented at the FEs for each of the destinations $d \in D$ produce Directed Acyclic Graphs (DAGs). In a properly functioning, stable network, each *forwarding DAG* spans all FEs in the network. There exists at least one directed path to an egress for each destination d from any FE in the network. In the simplest case of single-path routing and a destination with a single egress node, the forwarding DAG is an intree[7] rooted at the egress. In this paper, by *forwarding DAGs* we are always referring to the final stable set of forwarding decisions that FEs converge to after any network event. During transition from one stable set of forwarding DAGs to another, forwarding decisions may produce intermediate digraphs before converging to the final DAG(s). These intermediate digraphs may contain cycles, which are the graph representation of transient forwarding loops.

The analyses in this paper are valid for any routing scheme that selects acyclic paths to egress points. For simplicity of exposition, some of the examples assume single-path SPF, but none of the conclusions rely on that restriction.

For each destination d, a forwarding DAG $R(d)$ is constructed such that the vertices of $R(d)$ correspond to the nodes of the network $V(G)$.[8] For each vertex $v_i \in V(R(d))$ there is an arc from v_i to each of its next-hops for d.

[5]FE failures obviously make the failed node unreachable. We assume that the rest of the network remains connected.

[6]More generally, each destination may include additional constraints beyond IP address, *e.g.,* QoS parameters. So, for instance, a destination can refer to a destination IP address and a range of port numbers, accessed from a specified set of source addresses and port numbers.

[7]An *intree* is a directed tree, the root of which is reachable from any of the other vertices.

[8]R need be neither a tree nor use paths that are shortest by any metric. It is sufficient for R to contain at least one simple path to d from every node in $V(G)$, and no cycles.

Message complexity is a central concern for design of the Dissemination Plane. Dissemination serves as a transport service, so its primary consumption of routing system resources is through messaging. Further, message complexity serves as a convenient proxy for the aggregate overhead imposed by Dissemination. Each message represents the computation, storage, and communication necessary at the DE and throughout the CRN to produce, store, transmit, and consume forwarding information. In particular, we expect convergence time to be roughly propotional to message complexity. Because convergence is subject to factors such as network errors and retransmissions that are irrelevant to our main point, but unavoidable in any network analysis, we focus on message complexity as a general proxy.

In distributed routing such as a link-state IGP, messaging load is shared across all nodes of the network. But when the Decision plane is centralized, the DE is a bottleneck: every message in the Dissemination plane has the DE as either its source or its destination. In this case, message complexity at the DE is a critical consideration.

We assume that each dissemination message is addressed to a unique FE and contains one or more *destination updates*. Each destination update in turn contains the set of next-hops for a given destination. FEs process destination updates independently, in the order in which they are received.

3. SETTING AND PROBLEM STATEMENT

In a centralized routing system with a DE as a dissemination bottleneck, one key optimization of any protocol for the Dissemination plane is to deal with forwarding updates in terms of *groups* of destinations, rather than one destination (or egress) at a time. As an ultimate goal, for any given network event, we seek to send a single update message, covering all affected DAGs, to each FE. The reasoning behind this is straightforward. The DE would require $M = |D||V|$ messages to update each destination individually for each FE. More realistically, updates for all destinations sharing a common egress can be grouped together, for a smaller but still large $M = |V|^2$ number of messages.

That number would be reduced substantially if it were possible to send a single message to each FE with updates for all destinations. This improves the worst case to $M = |V|$. However, as we explain below, this optimization leads to transient forwarding loops.

[3] defines an ordering of FIB updates that guarantees loop-free convergence using only one message per FE for a single destination. Given a downed link $X \rightarrow Y$ and the cut reverse forwarding DAG $R_{X \rightarrow Y}(Y)$, the ordering criterion is that no FE in the DAG may update its FIB before its children. In other words, the ordering constraint is a topological sort of $R_{X \rightarrow Y}(Y)$. As long as this order is maintained for every updated destination, transient loops can never form.

Destination changes external to the network proper can also cause transient loops. Suppose, in Figure 1, destination a is reachable through FE 1 or FE 2. FE 4 then reaches destination a through FE 2. But FE 2 loses its path to destination a; now it is reachable only through FE 1, and FEs 2 and 4 must reroute. If FE 2 makes its FIB change before FE 4, then there is a one-hop forwarding loop between FE 2 and FE 4 for destination a. This can happen through an external network change (*e.g.,* learned from eBGP) or a policy

change (*e.g.,* to prefer the FE 1 link), but the point is that transient loops can still appear.

3.1 The No Ordering Condition

The message optimization goal is to bring all FEs up to date with the minimum number of messages. The smallest number of updates is obtained when the DE can update every FE with at most one message each, without producing transient loops. Each update message contains all destination modifications for the receiving FE. We call such an ordering of all destinations for all FEs a *global update order*. It is *not* always possible to find a global update order. In particular, we establish a necessary and sufficient condition for this to occur.

Transient forwarding loops will be possible only if there is a well-defined relationship between the forwarding DAGs before ($R(d)$) and after ($R'(d)$) the routing change:

THEOREM 1 (POTENTIAL LOOPS). *If the union of a priori and a posteriori forwarding DAGs:*

$$H(d) \triangleq R(d) \cup R'(d) \tag{1}$$

contains a cycle, then there exists at least one FIB update order which can produce a transient forwarding loop to the destination d.[9]

Let

$$C(d) \triangleq \text{cycles}(H(d)) \cap R'(d), \tag{2}$$

and let $\mathcal{L} = \bigcup_d C(d)$. Each $C(d)$ represents the FIB update order constraints due to an individual destination, so $\mathcal{L}$ is the global union of all constraints across all destinations and all failures. We then have the following theorem.

THEOREM 2 (NO ORDERING). *If $\mathcal{L}$ contains a directed cycle, then updating the forwarding tables (all entries at once) of FEs in any order will create a forwarding loop. Furthermore, if the No Ordering condition does not hold, then there exists a global update ordering of FEs that completely avoids forwarding loops.*

PROOF SKETCH. If $\mathcal{L}$ contains no directed cycles, then $\mathcal{L}$ is a directed acyclic graph (possibly consisting of multiple disconnected components). This DAG induces a partial ordering of the nodes (FEs). The DE can reach and update the forwarding tables of these nodes in an order that is compatible with the partial ordering induced by $\mathcal{L}$. This update order can never produce transient forwarding loops.

Conversely, if there *is* a directed cycle in $\mathcal{L}$, then there is no possible partial ordering. If a cycle includes nodes $v_i, v_j \in V(\mathcal{L})$, then there exist two directed paths $v_i \rightarrow \cdots \rightarrow v_j$ and $v_j \rightarrow \cdots \rightarrow v_i$. Therefore a loop-free update ordering requires both that v_i be updated before v_j and that v_j be updated before v_i. These two conditions cannot be simultaneously true, so there cannot be a loop-free global update ordering. □

The existence of the No Ordering condition may seem to be in contradiction with the ordering results of [3]. But No Ordering arises from the message minimization requirement of a CRN, and the goal to modify all destinations simultaneously. That goal is not a consideration in a distributed routing environment and so it does not arise in [3].

When the No Ordering condition does not hold, a preorder traversal of a topological sort of $\mathcal{L}$ provides a loop-free global updating ordering.

THEOREM 3. *If, for every directed arc $u \rightarrow v \in E(C)$ (that is, every arc in the graph $C(d)$, which is a proper subset of the arcs in G), the FIB of u is updated before the FIB of v, then there can be no transient loops to destination d in the network G.*

In general, it is not possible to update all forwarding DAGs in a single pass over the FEs. However, there are important classes of events for which a global update order always exists, that is, No Ordering does not hold.

THEOREM 4. *With any single link failure, there exists a loop-free global ordering.*

This is a straightforward corollary to the results of [3]. Given a simple network change involving the failure of a single link or node, or the activation of a single link or node, there exists an optimal update order that sends a single message to each affected FE. The message complexity, then, is $M = O(|V|) << O(|D||V|)$ in these cases. The algorithms in [3] do not encounter the No Ordering condition because they deal with state changes for each link individually. While this avoids both transient loops and No Ordering, in a CRN this approach would increase message complexity by a factor of ℓ for a failure of ℓ links.

A similar result applies to single-node failures.

THEOREM 5. *For any single-node failure, there exists a loop-free ordering of FEs.*

Also, events on a single destination, or group of destinations sharing a common forwarding DAG, cannot produce No Ordering. The proof in this case is simpler: if a single forwarding DAG has changed, then the FEs can be updated in a preorder traversal of the new DAG. There are, by definition of a DAG, no cycles, and so the ordering must exist.

Operational networks are, however, subject to a range of complex change events, as for example when multiple layer-2 links fail simultaneously because an underlying layer-1 resource is damaged, or when a burst of BGP updates affects multiple external destinations. Therefore the next question is, when do these more complex network events have the potential to produce the No Ordering condition?

3.2 Examples of No Ordering

What follows is a series of scenarios showing that plausible, uncomplicated network incidents can bring about No Ordering.

Multiple link failures. Consider the network on the left side of Figure 2. Paths are computed by SPF with link weights as shown in the figure. In particular, Node 5 reaches destination a by path $4, 1, 3$; Node 3 reaches destination b by path $1, 4, 5$.

If the links $1, 3$ and $4, 5$ were to fail simultaneously, the new network would be as in the right side of the figure. Node 5 now reaches destination a by path $1, 4, 5$, $H(a)$ (computed as shown in Equation 1) contains the directed cycle

[9]This theorem, along with most other theorems in this paper, is presented without proof due to space constraints. Proofs are available in the Technical Report version of this paper.

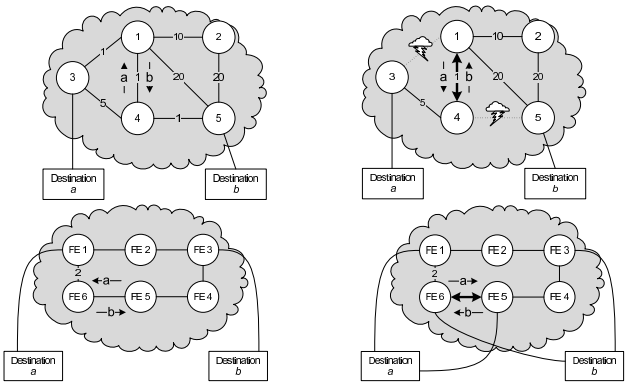

Figure 2: No Ordering appears (above) after two links fail and (below) from splitting of DAGs.

Network	Nodes	Links	Mean Degree	Diam.	Mean hops
GÉANT	25	39	3.12	3	2.79
AS 1755	137	280	4.09	7	5.46
AS 2914	573	1424	4.97	8	5.89
AS 3356	537	2304	8.58	6	4.22
AS 7018	534	1485	5.56	7	6.22

Table 1: Properties of sample networks.

		k-link failures				
Network	Sample	2	3	4	5	6
GÉANT	Full	8.0	20.0	32.0	44.0	56.3
AS 1755	1000	1.1	1.8	3.2	4.8	6.7
AS 2914	1000	0.1	0.4	1.5	2.9	2.9
AS 3356	$\geq$ 1000	0.02	0.1	0.1	0.3	0.8
AS 7118	1000	0.1	0.3	1.1	1.2	1.9

Table 2: Percentage of simulated multiple link failures that cause No Ordering

$1 \rightarrow 4 \rightarrow 1$, and $C(a) = 1 \rightarrow 4$, computed as shown in Equation 2. Node 3 now reaches destination b by path $4, 1, 5$, $H(b)$ contains the directed cycle $4 \rightarrow 1 \rightarrow 4$, and $C(b) = 4 \rightarrow 1$.

Now that $C(a) \cup C(b) \subseteq \mathcal{L}$ contains a cycle, $4 \rightarrow 1 \rightarrow 4$, the No Ordering condition holds. It is not possible to update this network with a single sequence of messages for both destinations. If Node 4 is updated first, then data intended for destination b from Node 1 will be forwarded to Node 4, which will forward them right back to 1. But if Node 1 is updated before Node 4, then data for destination a from Node 4 will be forwarded to Node 1, which will forward them back to Node 4. Either way, there is a transient loop.

Multiple DAGs splitting, merging, or uprooting. Consider a scenario involving multiple DAGs where simultaneously each of the DAGs splits into two or more DAGs, *i.e.,* the number of egress nodes through which a set of destinations are reached changes (as when BGP changes cause multiple egress nodes to become equally attractive to reach external destinations, and so internal nodes direct traffic to their closest egress nodes). The bottom half of Figure 2 shows initially two such DAGs, one for each of the destinations a and b. The DAGs for destinations a and b are trees rooted at nodes 1 and node 3 respectively. Node 5 reaches destination a through node 6, and 6 reaches b through 5. Then, an external routing change adds new egresses for the destinations: node 5 for destination a and node 6 for destination b. After this change, the DAG for destination a splits into two disjoint trees, one still rooted at node 1 and other at node 5. Similarly the DAG for destination b also splits into two, with one still rooted at node 3 and other at node 6. Again by inspecting the before and after DAGs, we see that the No Ordering condition has occurred. In the DAG before the split, node 5 forwards traffic to destination a through node 6, and node 6 forwards to destination b through node 5. After the split, node 6 forwards to destination a through node 5, and node 5 forwards to b through 6. As a result, if 5 is updated before 6, then data to destination b will be caught in a loop; but if 6 is updated before 5, then data to destination a will be in a loop.

A similar No Ordering condition will occur in this example if the egresses merge, that is, if the same event occurs in reverse. Destination a, initially reachable through nodes 1 and 5, loses connectivity through node 5, while destination b loses connectivity through node 6 and is only reachable through node 3.

No Ordering will also occur if the egresses move: *e.g.,* if the egress for destination a changes from node 1 to node 5, while the egress for destination b moves from node 3 to 6.

In summary, while several classes of network events always have a global ordering, there are also plausible scenarios in which No Ordering can occur. Since No Ordering *can* happen, the question for the next section is, how frequently *does* it happen in operational networks?

4. PROBABILITY OF NO ORDERING

In this section, motivated by our analytical findings, we quantify the prevalence and severity of the No Ordering condition during dissemination of routing updates. Both direct observation and simulations are used. First, we simulate multiple-link failures in large ISP networks. As shown in the previous section, these failures can in principle produce the No Ordering condition. The simulations will show the probability of producing No Ordering in realistic topologies. Second, we search BGP traces for evidence of egress tree changes. Again, the previous section showed that this can, in principle, produce the No Ordering condition, and so our first step is to assess through measurements the frequency of such occurrences in practice. The next step is then to determine how many of these egress tree changes actually give rise to the No Ordering condition. This assessment is carried out by way of simulations on inferred topologies for which we evaluate the impact of the egress tree changes derived from our measurements. The main outcome of these investigations is to establish that the No Ordering condition arises quite frequently during routing changes in actual networks. Hence, the frequency of transient loops during routing updates can also be high unless specific precautions are taken, as argued in this paper.

4.1 Multiple-Link Failures

One event that can produce No Ordering is the simultaneous failure of multiple links. To provide for a realistic setting when investigating the impact of multiple link failures, we use a number of different representative topologies derived from traces that provide a reasonable coverage of a

range of deployed topologies (see Table 1). These topologies are derived from two different sources. The first is based on traces of IS-IS link-state activity in the GÉANT European research network during September 2005. The remaining topologies come from the Rocketfuel project [12] and represent an inferred snapshot of the internal connectivity of four large-scale ISP networks as of 2003. These topologies are much larger and more richly connected than GÉANT, but the topologies are inferred rather than directly observed.

As mentioned earlier, the analysis of Section 3 is valid for arbitrary path selection schemes. However, for ease of illustration the simulations presented below assume an SPF scheme that selects a unique path from each FE to each egress node, using uniform weights for the bidirectional links gleaned from the topologies listed above, plus an arbitrary, stable tie-breaker for equal-cost paths. A limited subset of simulations were also run with equal-cost multi-path routing. Those results are presented in [11], and appear to indicate that the use of multipath routing significantly increases the incidence of No Ordering over the single-path results reported below.

Table 2 shows simulations of multiple-link failures for the topologies of Table 1. The first line shows the frequency of the No Ordering condition for all possible k-link failures in the GÉANT network for $2 \leq k \leq 6$. Over 8% of 2-link failures in GÉANT produced at least one instance of the No Ordering condition, and this frequency grows to close to 45% for 5-link failures. The No Ordering condition in a large, well-connected network is less common, as shown in the second part of Table 2. In larger networks, an exhaustive survey of multi-link failures is not combinatorially practical, and so the frequency of the No Ordering condition was evaluated from a representative sample of k simultaneous bidirectional link failures for $2 \leq k \leq 6$. Because routing behavior depends on which links are chosen for failure, 1000 distinct k-link failure scenarios were generated for each topology[10]. In all cases the frequency of No Ordering increases roughly with the number of links involved in the failure, so that, say, a fiber cut that affects multiple logical IP links is more likely to give rise to the No Ordering condition than the failure of a single router adapter.

Notice the two outlier networks in Table 2: GÉANT and AS 3356. We conjecture that the incidence of No Ordering is in part a function of link density. As the link density of a network (the ratio of links to nodes) increases, paths become shorter and the number of bottleneck links (which carry disproportionately many source-destination paths) decreases. Formation of No Ordering requires multiple paths, all simultaneously affected by a network event, sharing common links or nodes. Short, disjoint paths decrease the frequency of common links and nodes, and therefore decrease the incidence of No Ordering. This is reflected in the two outliers in Table 2: GÉANT, as a small network with low connectivity, suffers the most frequent No Ordering. AS 3356, with by far the highest link density in the sample, suffers the least frequent No Ordering.

4.2 Egress Changes Seen Through BGP

From our earlier discussion, we know that besides multiple link failures, another type of scenarios that can also give rise to the No Ordering condition are those that simultaneously affect several DAGs associated with destinations reachable through different egress points. In order to adequately assess the effect of such scenarios, it is necessary to first determine how common they are before trying to evaluate the extent to which they can produce the No Ordering condition.

Because BGP updates that affect the choice of exit point (BGP Next Hop attribute) are one of the common reasons behind egress DAG changes, we focus on extracting from BGP data the likelihood of changes that simultaneously affect multiple egress DAGs. The BGP data we use for that purpose is obtained from RouteView traces [10] gathered from seven ASes in April 2007. The seven ASes were selected because they provide *multiple vantage points* for the BGP updates that the AS experiences, and this is instrumental in allowing us estimate when those updates actually correspond to multiple egress DAG changes. Specifically, we look for BGP updates that affect a given prefix and that change egress point selection in the two vantage points[11]. A scenario where the two vantage points go from agreeing to disagreeing in their selection of egress points corresponds to a splitting of egress DAGs, and conversely one where they switch from distinct egress points to a common one corresponds to a merging of egress DAGs.

Because RouteViews traces are collected using eBGP from outside of the monitored AS, egress changes within the monitored AS are not visible. Instead, we infer Next Hop changes from the identity of the first hop AS specified in the AS Path attribute obtained from each vantage point. Specifically, we assume that the use of distinct first hop ASes by the two vantage points corresponds to a choice of different Next Hops and therefore egress points, and that a change in the first hop AS in the AS Path attribute advertised by a given vantage point corresponds to a switch to a different egress point by that vantage point. Based on this, we consider that a given egress DAG splits if the two vantage points go from advertising the same first hop AS in their respective AS Path attributes to advertising distinct ones. Conversely, we assume that the egress DAGs of the two vantage points merge if they switch to both advertising the same first hop AS in their respective AS Path attributes. We note that this approach entails some inaccuracies. In particular, the choice of the same first hop AS by two vantage points need not imply that they select the same egress point, *e.g.,* as when two ASes have multiple peering points and the IGP cost is used to select one. Conversely, a change in the first hop AS in the AS Path advertised by one of the vantage points need not always signal the selection of a new egress point, *e.g.,* because the two ASes are reachable through the same multi-homed egress router. The first type of inaccuracies can lead us to overlook some egress DAG splits, *i.e.,* when one egress point moves to a different router connected to the same AS, while the second type can produce false additional egress DAG splits, *i.e.,* when the egress point does not move in spite of a change in first hop AS. Nevertheless, we expect and have validated that these assumptions provide reasonable estimates of instances of egress DAGs merging and splitting.

The seven ASes with two vantage points that we selected

[10] For AS 3356, a sample of 1000 2-link failures found no incidences of No Ordering. A larger sample of 10^4 2-link failures found two instances.

[11] In order to eliminate transient changes that are simply created by iBGP latency between the two vantage points, as suggested in [13] we group all BGP updates taking place within a 120 sec time interval.

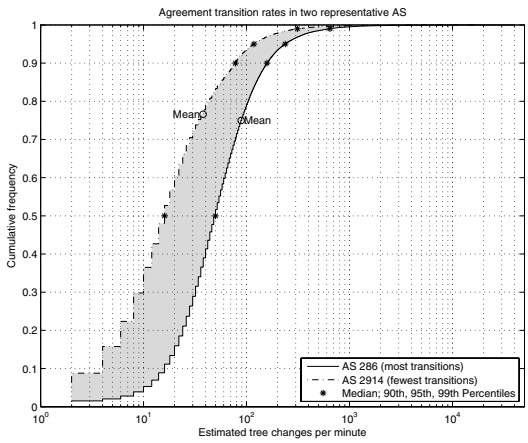

Figure 3: Incidence of changes in next-hop AS.

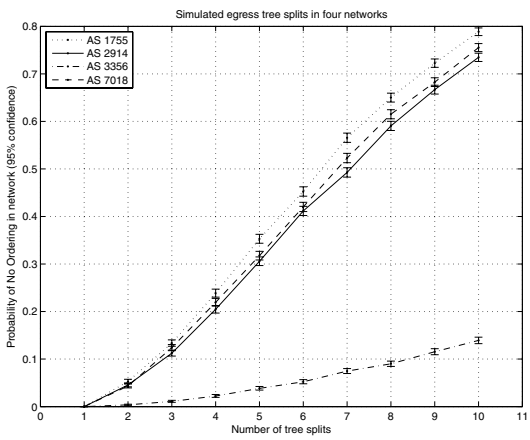

Figure 4: Effect of simulated simultaneous egress tree split events on the probability of a No Ordering condition anywhere in the network.

are ASes 286, 852, 2914, 3130, 3549, 6453, and 12682. Figure 3 shows the number of agreement transitions of prefixes taken from two of those seven ASes (AS 286 and AS 2914) during April 2007. These two AS are of interest because they are, respectively, the most and least active among the seven visible from RouteViews: that is, the distributions for the other four AS fall somewhere in between the two curves plotted in Figure 3. In all cases, agreement transitions are frequent in the data, with the median lying between 8 and 25 per 2-minute interval, depending upon the AS. This means that even in the AS with the fewest agreement transitions, half the 120-second windows during the month of April 2007 had eight or more of them.

4.3 No Ordering in Egress Splits

Having established the frequency of tree merges and splits in actual networks, the next step is to quantify how often they give rise to No Ordering. The focus will be on egress tree splits whose impact will be evaluated by simulation using the inferred Rocketfuel topologies of Table 1.

The simulation procedure for an egress tree split is as follows. Each simulated split begins with a selection at random of two nodes, X and Y. Before the event, all nodes in the network route to a destination d through X as the egress. After the event, the network is partitioned between nodes that reach d through egresses X and Y. For this simulation, the partition is decided by the shortest path: each node routes to the egress that is the fewest hops away. Ties are broken in favor of X, in an attempt to minimize thrashing. Only nodes that have fewer hops to Y than to X need their forwarding tables changed.

Each simulated event involves two or more tree splits, involving two distinct sets of egress nodes chosen at random, using all nodes in the network as potential egress points with equal probability.

To minimize sampling error, each scenario (selected AS and number of trees being split - varied from 2 to 9) was repeated 10^4 times. For all measurements we record the 95% confidence intervals, as summarized in Figure 4. The No Ordering condition is a frequent occurrence in these simulations. The frequency with which it appears increases approximately linearly with the number of trees being split across the entire range observed, with AS 3356 again an outlier because of its higher link density as conjectured earlier.

As for the multi-link failure scenarios of Section 4.1, the simulations presented here assume single-path routing, but results when multiple equal cost paths are allowed can be found in [11] and again point to an *increases* in the incidence of No Ordering. Thus, given the common use of multipath routing in practice, one can expect an even higher frequency of occurrence of No Ordering than reported in this paper.

The main conclusion is that given the frequency with which multiple egress trees split in operational networks (as measured in Section 4.2), and the frequency with these splits can produce No Ordering (as simulated in this section), it is reasonable to conclude that No Ordering occurs frequently in operational networks. Each instance of No Ordering *must* produce transient forwarding loops if the DE attempts to update all FEs in a single global order. This adds up to significant network disruption.

		Message complexity (percentiles)			Greedy factor (percentiles)		
Net	Fail	10	50	90	10	50	90
GÉANT	link	6	20	72	2.4	3.7	5.4
	node	48	64	160	1.9	2.6	6.4
AS 1755	link	112	302	677	2.3	4.0	7.7
	node	534	907	1983	3.9	6.6	14.
AS 2914	link	502	1873	4864	2.2	5.2	11.
	node	2621	5731	14535	4.6	10.	25.
AS 3356	link	235	833	2112	1.6	2.5	5.2
	node	2024	4549	10778	3.8	8.5	20.
AS 7018	link	351	1323	5726	1.6	3.9	14.
	node	2071	6640	18709	3.9	12.	35.

Table 3: Message complexity of per-destination updates.

5. MESSAGE OPTIMIZATION

Because of the No Ordering condition, it is not always possible to find a global update order to prevent transient forwarding loops. It is natural to ask next if it is possible to partition destinations so that within each partition there is a valid, loop-free ordering of FEs for update? That is, given a set D of forwarding DAGs, each of which needs to be modified in one or more FEs, is it possible to form a partition $\{D_i\}$ of D such that for every D_i the No Ordering

condition does not hold? The answer is Yes: as established in Section 3.1, the No Ordering condition cannot occur when updating a single DAG. If each of the partitions D_i contains a single DAG, then that partition must have a loop-free update order. So it is possible to avoid transient loops for any network event by sending a separate update to each FE for each affected DAG. The DE can then calculate an independent, loop-free update order for for each DAG.

To understand the cost of this naïve approach, consider its message complexity.

To tally the message complexity in any network event, we first define the condition $\delta_{uv}(D, D')$ to indicate that when a network event changes the forwarding DAGs from D to D', the set of downstream neighbors of node u changes for destination v. In other words, the nexthop set for node u needs to be updated for destination v. (For brevity we abbreviate this as δ_{uv}, leaving the DAGs implicit.) Message complexity depends on the number of forwarding DAGs affected by a given network event, as well as the number of affected upstream nodes[12] in the corresponding forwarding DAGs[13]. For naïve updates, we assume that it takes exactly one message to inform a single forwarding node of a change to each DAG. Their message complexity is computed thusly:

$$M_0(G, D, D') = \sum_{u \in V(G)} \sum_{v \in D'} \delta_{uv}. \tag{3}$$

Message complexity when a global update order is available is simpler, with each node requiring at most one message if any of its next-hops have changed:

$$M_1(G, D, D') = \sum_{u \in V(G)} \mathbf{1}_{(\sum_{v \in D'} \delta_{uv})>0}. \tag{4}$$

Individual updates. The middle columns of Table 3, labeled "Message complexity," show the M_0 message complexity (*i.e.,* without update grouping) of all single-link and single-node failures, as simulated for each of the listed networks. The message complexity varies depending on the particular network event that triggers the update, so the table shows median, tenth, and ninetieth percentiles of the results.

Remember that, by Theorems 4 and 5, none of these events can produce the No Ordering condition; so there must be a global update order available, although naïve update, by definition does not use it. For all of the large networks, even these simple network events require hundreds or thousands of messages to disseminate updates.

Update grouping. Having established that message minimization is a worthwhile goal in a CRN, but that dissemination with a single message per FE leads to transient loops, it is time to seek a realizable compromise. To improve message efficiency, we seek to consolidate DAGs into K update groups, such that $1 \leq K \leq |D|$. Compared with a naïve per destination update mechanism, this could reduce message complexity from $|D||V|$ down to $K|V|$, yielding a significant improvement if $K << |D|$. Unfortunately, the following theorem states that the problem of finding the minimum set of K update groups is NP-hard. The proof of the theorem is in [11].

[12] Assuming that as u, they need to update their next hop(s) for destination v.

[13] This neglects such low-level details as retransmissions and segmentation of large update messages.

```
GREEDY(D, D')
 1  B ← ∅
 2  for d ∈ D and d' ∈ D'
 3        do L ← cycles(d ∪ d') ∩ d'
 4           f ← TRUE
 5           for b ∈ B
 6                 do if cycles(b ∪ L) = ∅
 7                       then b ← b ∪ L
 8                            f ← FALSE
 9                            break
10           if f
11              then B ← B ∪ {L}
12  return B
```

Figure 5: Greedy Destination Partition Algorithm

THEOREM 6 (DESTINATION PARTITION). *The problem of partitioning destinations into the minimal number of update groups, so that for each group there exists an order of node (FE) updates that allows the simultaneous update of the forwarding states of all destinations while avoiding the creation of forwarding loops is NP-Hard.*

The proof, available in the technical report [11], relies on a reduction from the NP-Complete Independent Set problem to the Destination Partition problem.

5.1 Greedy Destination Partition Algorithm

Not only is Destination Partition NP-Hard, but we conjecture, based on its relationship to the Independent Set problem, that it is also difficult to approximate. Independent Set belongs to a class of problems that have defied tight approximation [9]. In fact, a simple, fast greedy approximation is the best solution available, using polynomial processing time to construct update groups which are guaranteed to be loop-free but which may not be a minimum partition. This algorithm is called in the DE after one or more network events has forced a recalculation of forwarding paths. Given the network-wide sets of forwarding DAGs from before and after the events, the algorithm assigns each DAG to an update group and calculates an update order for the affected FEs in each group. The plan of the algorithm is simple: it maintains a set of update groups, B. For each destination DAG d that requires an update, the algorithm considers adding the new DAG to each update group in turn. The DAG d is assigned to the first update group $b \in B$ for which d does not produce the No Ordering condition.

Figure 5 describes the greedy algorithm in more detail. Inputs are D, the initial set of forwarding DAGs before the network change; and D', the set of DAGs changed by the network event. The result, B, is a set of destination groups. Each group $b \in B$ comprises a set of destinations and a combined DAG that represents the update ordering constraints on the group of destinations. We assume the availability of an auxiliary function *cycles*, which returns the subgraph of its argument comprising all arcs that are part of directed cycles.

The algorithm starts with an empty result set $B = \emptyset$, then at line 2 considers each forwarding DAG both before (d) and after (d') the network event. At line 3, the algorithm

calculates L, the update order constraints due to d. Then, for each update group $b \in B$, the algorithm checks for cycles in $b \cup L$. If there are no cycles, then adding d to b does not produce the No Ordering condition, and so the algorithm adds d to b and skips to the next forwarding DAG. If d would produce No Ordering in group b, the algorithm moves to the next update group. Finally, if d produces No Ordering in all of the current update groups, the algorithm creates a new update group for d at line 10.

For each destination group, the associated DAG in B constrains the set of loop-free update orders: if $uv \in E(b \in B)$, then v must be updated before u. No update order that obeys these constraints can produce a transient forwarding loop. That leaves the DE free to choose a conforming update order to minimize network convergence time and maximize message parallelism.

The worst-case time complexity of the greedy algorithm is $O(|D|^2|V|^2)$, because finding a group for each destination takes $O(|D|^2|V|)$. In practice, its performance is much better, because the number of groups is small. Also, the algorithm permits optimization by exploiting the flexibility in both the selection of destinations on line 2 and update groups in line 5.

Note: *in the absence of the No Ordering condition, the greedy algorithm will find a global update order that does not produce transient loops.* For network events that give rise to No Ordering, the outcome of the greedy algorithm depends on the order in which it considers destinations in the loop beginning on line 2. In the worst case, the greedy algorithm produces $\min(|L|, |D|)$ update groups.[14]

5.2 Evaluation of the Greedy Algorithm

This section evaluates the effectiveness of the Greedy algorithm of Section 5.1 in identifying an efficient update ordering that guarantees the absence of transient loops with approximately minimum messages. We use two related metrics for this measurement. First is raw message complexity as defined in Section 5. Also, because the Greedy Algorithm does not directly optimize message complexity, but rather seeks to minimize the number of update groups, we use number of update groups as a secondary metric.

Greedy Algorithm - Global Ordering. After execution of the greedy algorithm, the DE must send one message per update group to each FE affected by the egress nodes in the group. In the best case, when No Ordering is absent, the Greedy Algorithm produces a single update group and the resulting message complexity is no more than the number of FEs in the network.[15] This is true in the majority of real network failures, and is true of *all* of the single-link and single-node failures simulated in Table 3. The rightmost column of Table 3, labeled "Greedy factor," shows the improvement in message complexity obtained by sending a single update per FE, using an order found by the Greedy Algorithm. The computed values correspond to the tenth, median and ninetieth percentiles of the ratio M_0/M_1. Reductions by factors of 5 or more are common in some cases can be several orders of magnitude.

[14] Also, note that we need only enforce a *partial* order of FEs within each update group. The DE is free to apply additional ordering constraints for other purposes, *e.g.*, to minimize the duration of black holes caused by link outages.

[15] Remember that the loop-free update order is constrained; an arbitrary update ordering runs the risk of producing transient forwarding loops. The Greedy algorithm produces a loop-free global ordering, if one exists.

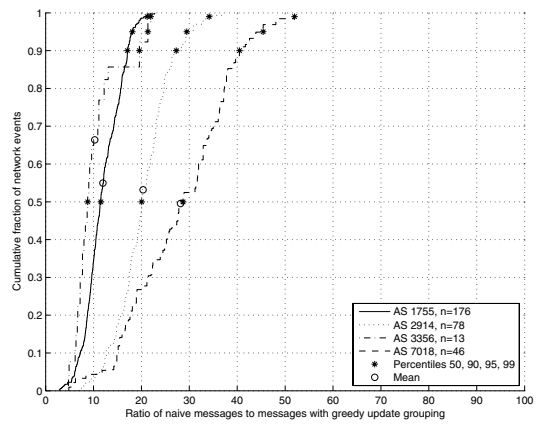

Figure 6: Improvement in message complexity after Greedy update grouping of k-link failures.

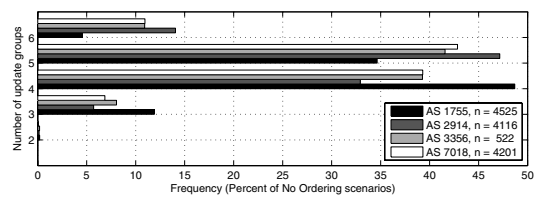

Figure 7: Number of update groups found by Greedy Algorithm in scenarios with six DAGs split

In more complex network events, the reduction of message complexity is even more pronounced. Starting with the simulations reported in Table 2, we concentrated on the subset of cases that did not produce No Ordering. In each case, the Greedy destination partition algorithm found a single update group with a global update order. The resulting message complexity reductions, again corresponding to M_0/M_1, are almost always at least a factor 5, and in rare cases can be nearly a factor 100.

Greedy Algorithm - No Ordering. Next, we consider scenarios where No Ordering is present. First, we generalize Equation 4 to measure message complexity with a set B of update groups:

$$M_B(G, D, B) = \sum_{b \in B} \sum_{u \in G} \mathbf{1}_{(\sum_{v \in b} \delta_{uv})>0}. \qquad (5)$$

Here we consider the subset of k-link failure results from Table 2 that *do* produce No Ordering. Because results of the Greedy algorithm depend in part on the order in which nodes are considered for grouping, for each network event we use 100 runs of the algorithm, each considering the nodes in a distinct order. The resulting message complexity reductions, M_0/M_B, shown in Figure 6, are very similar to the results for M_0/M_1, showing that the efficacy of the Greedy algorithm is insensitive to the presence of No Ordering and the number of update groups $|B|$.

DAG-split scenarios, as illustrated in Figure 4 and described in Section 4.3, also produce No Ordering, but with

differences from k-link failures. Empirically, No Ordering occurs more frequently with DAG splits than with link failures. But while each link failure affects every DAG that includes the failed link, each DAG split affects only one DAG.

Because the number of affected DAGs is limited to k, the number of split DAGs, for small k it is practical to apply the Greedy Algorithm exhaustively with all $k!$ permutations (in Figure 5 on line 2). To study the Greedy Algorithm in DAG-split scenarios with the No Ordering condition, we took each split of $k \leq 6$ DAGs in Figure 4 that produced No Ordering, and ran the Greedy Algorithm with all permutations. The first result was that DAG permutation had no effect on the number of update groups found by the algorithm. While this result is surprising, it is also encouraging, because it means that in these cases the Greedy Algorithm always found the optimal number of update groups. However, this property is not proven for the general case of DAG splits, and is obviously not the case for general network events, as shown by the k-link failure results.

Figure 7 shows the number of update groups found by the Greedy Algorithm[16] across four topologies with six split DAGs. The results are better than the naïve approach, using six update groups with one DAG per group, approximately 90% of the time. This conclusion is consistent across all four networks.

Thanks to the small number of update groups, the Greedy algorithm runs quickly: the fewer groups, the fewer graphs the algorithm must test for cycles.

6. CONCLUSIONS

This paper demonstrates that the problem of avoiding transient routing loops changes significantly in a centralized routing environment as compared with a decentralized IGP. Centralized routing introduces a bottleneck in the Distribution plane at the node(s) providing the centralized Decision service. This in turn calls for minimizing the number of messages they originate.

As a result, message efficiency is a key consideration in the operation of a DE seeking to update its FEs after a network change while avoiding transient routing loops. In general, changes affect multiple destinations and FEs at a time. The ideal in terms of message complexity is to send a single update message to each affected FE. This is not always possible without creating loops due to a condition called No Ordering, and the paper provided evidences that this condition commonly arises in operational networks. We then turned to explore the extent to which it was possible to reduce message complexity by grouping updates whenever possible. It first showed that identifying an optimal solution was NP-hard, and then proceeded to propose and evaluate a greedy approximation that was found to significantly improve message efficiency. The paper and its results can help inform and realize the design and deployment of centralized routing solutions.

7. REFERENCES

[1] Albrightson, R., Garcia-Luna-Aceves, J., and Boyle, J. EIGRP–A fast routing protocol based on distance vectors. In *Proc. Network/Interop* (Las Vegas, NV, May 1994).

[2] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and van der Merwe, K. The case for separating routing from routers. In *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)* (Portland, OR, September 2004).

[3] Francois, P., and Bonaventure, O. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Transactions on Networking 15*, 6 (Dec. 2007), 1280–1292.

[4] Francois, P., Bonaventure, O., Shand, M., Bryant, S., and Privedi, S. Loop-free convergence using oFIB. Internet draft, February 2008. Work in progress, revision 02.

[5] Fu, J., Sjödin, P., and Karlsson, G. Loop-free updates of forwarding tables. Tech. rep., Royal Institute of Technology (KTH), Aug. 2007. Accepted in IEEE Transactions on Network and Service Management (TNSM).

[6] Garcia-Lunes-Aceves, J. J. Loop-free routing using diffusing computations. *IEEE/ACM Trans. Netw. 1*, 1 (February 1993), 130–141.

[7] Greenberg, A., Hjalmtysson, G., Maltz, D., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A clean slate 4D approach to network control and management. *ACM Comput. Commun. Rev. (CCR) 35*, 5 (October 2005), 41–54.

[8] Hengartner, U., Moon, S., Mortier, R., and Diot, C. Detection and analysis of routing loops in packet traces. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (New York, NY, USA, 2002), ACM Press, pp. 107–112.

[9] Hochbaum, D. S. Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, Ed. PWS Publishing, 1997, ch. 3, pp. 94–143.

[10] Meyer, D., et al. University of Oregon Route Views project. `http://www.routeviews.org`.

[11] Peterson, H., Sen, S., Chandrashekhar, J., Gao, L., Guerin, R., and Zhang, Z.-L. Message-efficient dissemination for loop-free centralized routing. Tech. rep., University of Minnesota Department of Computer Science, 2008. `http://www-users.cs.umn.edu/~peterson/tr-dddd-2008-01.pdf`.

[12] Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking 12*, 1 (2004), 2–16.

[13] Teixeira, R., Shaikh, A., Griffin, T., and Rexford, J. Dynamics of hot-potato routing in IP networks. *ACM SIGMETRICS Performance Evaluation Review 32*, 1 (2004), 307–319.

[14] Yan, H., Maltz, D. A., Ng, T. S. E., Gogineni, H., Zhang, H., and Cai, Z. Tesseract: A 4D network control plane. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)* (2007).

[16] GÉANT was not included because its clients are not multihomed and the network is so small that six simultaneously split DAGs is not a plausible scenario.