Inference and Labeling of Metric-Induced Network Topologies*

Azer Bestavros best@cs.bu.edu
John Byers byers@cs.bu.edu
Khaled Harfoush harfoush@cs.bu.edu

Computer Science Department
Boston University
Boston, MA 02215

Technical Report
BUCS-TR-2001-010

May 2001

Abstract

The development and deployment of distributed network-aware applications and services over the Internet require the ability to compile and maintain a model of the underlying network resources with respect to (one or more) characteristic properties of interest. To be manageable, such models must be compact, and must enable a representation of properties along temporal, spatial, and measurement resolution dimensions. In this paper, we propose a general framework for the construction of such metric-induced models using end-to-end measurements.

We instantiate our approach using one such property, packet loss rates, and present an analytical framework for the characterization of Internet *loss topologies*. From the perspective of a server the loss topology is a logical tree rooted at the server with clients at its leaves, in which edges represent lossy paths between a pair of internal network nodes.

We show how end-to-end unicast packet probing techniques could be used to (1) infer a loss topology and (2) identify the loss rates of links in an existing loss topology. Correct, efficient inference of loss topology information enables new techniques for aggregate congestion control, QoS admission control, connection scheduling and mirror site selection. We report on simulation, implementation, and Internet deployment results that show the effectiveness of our approach and its robustness in terms of its accuracy and convergence over a wide range of network conditions.

Keywords: End-to-end measurement; Packet-pair probing; Bayesian Probing; TCP/IP; Internet Tomography; Performance evaluation.

*This work was partially supported by NSF research grants CCR-9706685 and ANIR-9986397.

1 Introduction

A central challenge underlying the design of a network transport protocol is that it use shared network resources efficiently, both to fairly accommodate other flows and to satisfy the specific requirements of applications. To do so, transport protocols must necessarily acquire information about network conditions. However, today's Internet does not normally supply such information explicitly, and when it does (such as when ECN [35] is used), the information provided may be of limited utility.

A large and growing body of literature considers remedies to this problem which center around augmentation of the existing Internet infrastructure to accommodate applications with specific quality-of-service (QoS) requirements. The approach we advocate instead, is to allow endpoints to perform statistical end-to-end observations along connections and to *correlate these observations across connections* to build up a detailed, multiscale understanding representation of network conditions of interest. While this passive approach to inferring network conditions ultimately provides much less power to applications than a model in which statistical or guaranteed levels of service are provided, it has the appealing property that it can be deployed in today's Internet.

As a starting point, it is already well known that the end-to-end measurements made in the course of normal operations by most transport protocols provide a wealth of information about the end-to-end characteristics of a path in the network. For example, end-to-end bottleneck bandwidth rates, round-trip times and packet loss statistics can all be inferred from the dynamics of a TCP connection [1]. In addition to the above connection-specific statistics, we show that end-to-end measurements correlated across different connections can be used to infer conditions at the level of individual links. Moreover, we present a general methodology for incorporating and combining sets of end-to-end measurements at different servers in order to construct a compact model of the network.

Motivation: Many of today's Internet services are administered through multiple co-location facilities that are distributed over the wide area. Examples include Content Distribution Networks (CDNs) and Application Service Providers (ASPs). Internet servers at such facilities (including Web servers, proxy servers, content distribution outlets, streaming media servers, etc.) may potentially command a large number of concurrent unicast connections. As such, these servers are likely to contribute a significant portion of the data traffic on the Internet.[1] We use the term *Mass Servers* to refer to this class of Internet servers—namely, *Mass*ively Accessed *Servers* that command a large number of concurrent unicast connections. A collection of Mass servers distributed over a WAN is in a unique position to infer valuable network state information that could be effectively used to maximize resource utilization and optimize content delivery and distribution. In order for such optimization strategies to be practical, an endpoint must be able to quickly and accurately infer the internal characteristics of the network connecting it to other endpoints. These characteristics are measured in terms of one or more metrics of interest such as hop-count, loss rate, capacity, delay, and jitter.

Previous Internet characterization studies have focused on the discovery of characteristics of Internet structures that are tightly related to physical attributes of the network such as buffer capacities, link speeds or the AS topology [14]. For the real-time resource management problems that face Mass servers, an accurate characterization of the physical resources between the server and its clients is not necessary.

[1] As anecdotal evidence, recent analysis of campus traffic showed that 3% of all accessed Internet servers were responsible for over 50% of all flows and that 0.1% (only 6 servers) were responsible for over 10% of all flows.

Rather, an abstraction that captures the shared resources to be judiciously managed is sufficient. In the case of Mass servers, the key resources which must be managed are those which affect the perceived performance of applications deployed at these Mass servers. For a streaming media delivery application, a model of the network that captures jittery paths may be used for server selection purposes. Alternatively, for a distributed caching or content distribution application, a model of the network that captures lossy or congested paths may be used for optimizing replica placement.

Metric-Induced Network Topologies (MINT): Given a set of network endpoints, we define a *Metric-Induced Network Topology* (MINT) to be a weighted, directed graph. The vertices of this graph represent network endpoints as well as routers, while an edge in this graph represents a *sequence* of one or more physical network links. The weight on each edge correspond to a real-valued quantity that a specific metric function attributes to the sequence of physical links represented by the edge. Only those sequences of links which collectively have a metric value above a parameterized threshold $c \geq 0$ are represented by edges in the topology.

To illustrate the MINT framework, which we define formally in Section 3, let packet loss rate be the metric of interest. The topology induced by this metric (i.e. the "loss topology") is a graph which can be obtained by merely collapsing the physical topology graph connecting the set of endpoints—by reducing any sequence of links with insignificant loss rates in the physical topology graph to an internal node in the loss topology graph.

Paper Contributions and Overview: This paper introduces the concept of Metric-Induced Network Topologies (MINT) and proposes general procedures for MINT inference and labeling and manipulation over multiple scales along temporal, spatial, and measurement resolution dimensions. We define a broad class of metrics for which our proposed techniques are applicable. We instantiate MINT for one such metric, packet loss rates, by showing that recently proposed end-to-end techniques for the estimation of shared losses could be leveraged to enable the characterization of *loss topologies*. We present results of extensive simulations that demonstrate the accuracy and convergence of our loss topology inference and labeling techniques. In addition, we present an implementation of our loss topology identification techniques, which are available to server-side applications through a Linux API called Periscope (a Probing Engine for the Recovery of Internet Subgraphs). Finally, we present results we obtained by using Periscope to infer and label actual Internet loss topologies.

The remainder of this paper is organized as follows. In Section 2, we review existing literature and related work. In Section 3, we present our basic framework for metric-induced topology identification. In Section 4, we instantiate this framework for the problem of characterizing *loss topologies*. In Section 5, we describe results of extensive simulation experiments that we have conducted to evaluate this approach. In Section 6, we report on implementation and Internet deployment results of Periscope. We conclude this paper in Section 7 with a summary and a description of our ongoing research.

2 Related Work

Inference and prediction of network conditions is of fundamental importance to a range of network-aware applications including server selection [4, 8, 15], cache management [7, 38] and replica placement [20, 10, 40]. Interest in these applications has spawned numerous research efforts now underway in the area of inference and prediction. In this section, we start with a general taxonomy of network characterization efforts in the context of our current work. Following that general overview, we focus on the most relevant of these efforts to the work presented in this paper.

	Network Measurement		End-to-End Measurement			
	Active	**Passive**	**Active**		**Passive**	
Multicast	[27]	[22]		[6, 5, 36, 41]		
Unicast	[23]	[22]	[37, 21, 11, 13]	[32, 41, 37, 9]	[37, 21, 26]	[37]
			Sender	**Receiver**	**Sender**	**Receiver**

Table 1: A Taxonomy of Network Characterization Efforts (with Representative Citations).

A Taxonomy: One widely adopted strategy for the characterization of network properties/conditions is to analyze data collected from network-internal resources (e.g. router ICMP replies, BGP routing tables) to generate performance reports [22, 27, 23] or to perform Internet topology characterization [19, 17, 29]. This approach is best applied over long-time scales to produce aggregated analyses, but does not lend itself well to providing answers to the fine-grained needs of network-aware applications.

Another approach is statistical inference of network internal characteristics based on end-to-end measurements of point-to-point traffic [3, 9, 6, 41, 24, 36, 32, 28, 39]. This approach can be further classified by *active* approaches and *passive* approaches. The benefit of the former is flexibility: one can make measurements at those locations and times which are most valuable; while the benefit of the latter is that no additional bandwidth and network resources are consumed solely for the purpose of data collection.

Cutting across other dimensions, one can also classify end-to-end approaches as either receiver-oriented or sender-oriented, depending on where inferences are made; and multicast-driven or unicast-driven, depending on the model used to transmit probe traffic. Table 1 illustrates this taxonomy with references to studies and projects that fall within each of its different categories. The framework presented in this paper falls into the general category of end-to-end measurements using either passive or active probing.[2] It is sender-based and is targeted for unicast environments.

Estimation of Network Conditions Using End-to-End Measurements: The specific problem of identifying and characterizing loss topologies is motivated in part by recent work on topological inference over *multicast* sessions [5, 12, 36]. By making purely end-to-end observations of packet loss at endpoints of multicast sessions, Ratnasamy and McCanne [36] and Cáceres *et al.* [5] have demonstrated how to make unbiased, maximum likelihood estimation inferences of (a) the multicast tree topology and (b) the packet loss rates on the edges of the tree, respectively.

Shared Loss Measurement: At the core of our methodology for constructing loss topologies is the need for a procedure for the measurement of shared losses between one source and multiple destinations. A number of recent efforts have addressed this problem; all would be suitable as underlying procedures in our framework.

In [37], Rubenstein, Kurose, and Towsley propose the use of end-to-end probing to detect shared points of congestion (POCs). By their definition, a point of congestion is shared when a set of routers are dropping and/or delaying packets from both flows. Their probing technique for identifying POCs uses a Poisson process to generate probe traffic to a pair of remote endpoints and computes cross-correlation measures between pairs of packets from these flows. In [21], Harfoush, Bestavros and Byers present an alternative technique for the identification of shared losses using a Bayesian Probing (BP) approach. Their

[2] Our PERISCOPE tool uses active probing due to the nature of the underlying technique it employs to measure shared losses.

BP approach relies on using of "packet-pair" techniques, originally used by Keshav [24], Carter and Crovella [9] and others [32, 31], to determine bottleneck bandwidth on a network path. Packet-pair probes are sent to a pair of *different* receivers to introduce loss and delay correlation, much the same way a multicast packet to these two receivers introduces correlation.

A number of recent efforts [13, 11, 25, 26] have generalized the notion of "packet pairs" probing. For example, the striped unicast probes of Duffield, LoPresti, Paxson, and Towsley uses a sequence of back-to-back packets sent to different receivers as an approximation of a multicast probe, thus enabling them to use link loss and delay inference techniques devised for multicast probes [12].

Internet Characterization Tools and Services: A number of research groups have generated maps of the Internet using route tracing tools such as `traceroute` [33, 16]. Work led by Govindan [34, 18] outlines heuristic techniques for generating complete domain maps. Pansiot and Grad [29] reported on and characterized the topology resulting from a detailed aggregation of end-to-end routes collected in 1995. Paxson [30, 31] deployed a "network probe daemon" (NPD) at 37 sites in the wide-area, which used `traceroute` to investigate end-to-end routing behavior and later, studied performance of transport protocols between all pairs of sites over several weeks.

3 Metric-Induced Network Topology Identification

In this section, we describe the MINT framework for the characterization of metric-induced topologies.

3.1 Summarizing Shared Network Resources using MINT

As motivated in the introduction, for many applications, it is often convenient to cluster clients of a Mass Server according to the extent to which they consume a shared network resource. Such sharing is often quantified by one of a number of *metrics*, including shared jitter, shared network delay or shared packet loss.

Depending on the resource of interest, different *metrics* would naturally be used to measure the extent of sharing that is taking place, or the QoS to be expected of the resource. For example, in the context of content delivery with real-time constraints, network delay and jitter are two metrics which are used to evaluate the performance perceived by a client. Similarly, a server may wish to classify or cluster those clients which lie beyond a link with long delay or a link with high jitter so as to provide sufficient support to deliver a specific level of service to those clients.

The labeled topology from the server to the clients with respect to a given performance metric is a *Metric-Induced Network Topology* (MINT). In many instances, edges with small labels, representing negligible, or statistically insignificant amounts of delay, jitter or loss, can safely be disregarded. Therefore, effective methods for producing a metric-induced topology must be *sensitive* to different possible thresholds for what constitutes an observable value of the metric. This is one sense in which the framework we present in this Section, and the Periscope tool we present in Section 6 provide, a *multiscale representation* of a metric-induced topology. Beginning with definitions presented in [5], we formalize these notions below:

Consider the set of links used to route unicast traffic between a server and a set of clients. Together these links form a tree $\mathcal{T} = (V, E)$ rooted at the server, with the clients at the leaves and routers at the internal nodes. The flows of packets sent from a server to an arbitrary subset of its clients share some (possibly none) of $\mathcal{T}$'s links and eventually diverge on separate links en route to the different clients.

Definition 1 *The* physical topology *connecting a server to a set of clients is the tree $\mathcal{T}$ induced by IP routing with the server at the root, clients at the leaves and routers at the internal nodes of the tree.*

The following operation is needed to help us define that portion of the physical topology observable by end-to-end measurements. We say that we *collapse* a node i of a tree into its parent j if we delete edge (i, j) from the tree and attach all children of i to j, by replacing all edges (i, k) with edges (j, k) for all $k \neq j$. Note that when this operation is applied to a tree, the resulting topology is also a tree.

Definition 2 *The* logical topology *induced by a physical topology $\mathcal{T}$ is the tree formed from $\mathcal{T}$ after all internal nodes with only one child have been collapsed into their parent recursively.*

The logical topology reflects the topology made visible by end-to-end measurements, as end-to-end techniques will be unable to assign metrical labels on a link-by-link basis to a sequence of physical links in a chain with no branching.

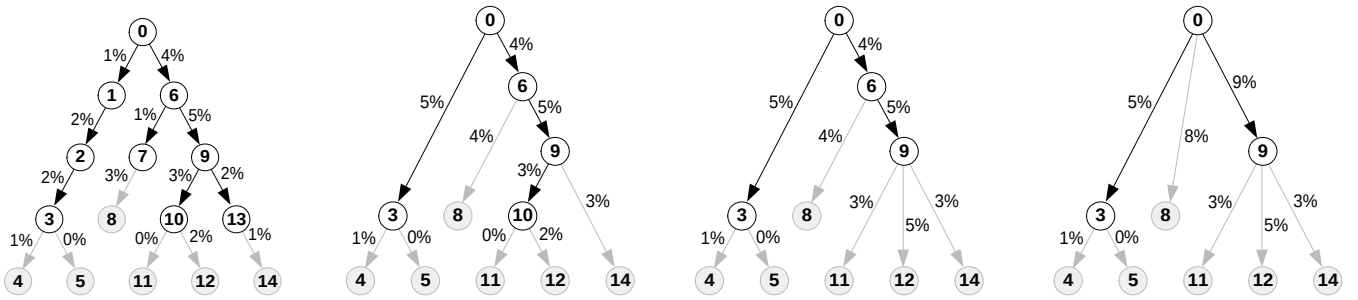
We now extend our definitions to apply to topologies with edges labeled according to a metric. For the purpose of our discussion, we define a *metric* to be a function f whose domain is the set of paths in a tree (or set of simple paths in a graph) and whose range is the reals. We refer to a *labeled topology* as any topology in which values of the metric have been applied to each link in the topology, noting that links may correspond to multi-hop physical paths as well as physical links. For some metrics, end-to-end observations will have difficulty distinguishing a small metrical value ϵ from zero and as a result, may misclassify incidence of sharing. For this reason, we suggest parameterizing any labeling algorithm with a sensitivity parameter c which is the minimum value of a label which can be applied to any internal link in the resulting topology $\mathcal{T}_c$. Our definition of metric-induced topology incorporates such a parameter:

Definition 3 *A metric-induced topology with sensitivity parameter c is the labelled topology formed when all internal nodes i in a logical topology whose parent link L_i has a value $f(L_i) \leq c$ have been collapsed into their parent.*[3]

Based on these definitions, an edge in a MINT graph represents a sequence of one or more physical network hops that collectively exhibit observable values of that metric. Clearly, the larger the value of the sensitivity parameter c, the smaller the number of links present in the topology $\mathcal{T}_c$. In this sense, $\mathcal{T}_c$ represents a "condensed" version of the original topology, and increasing the value of the sensitivity parameter c has the effect of increasing the level of condensation. We refer to the metric-induced topology for which $c = 0$ (i.e. $\mathcal{T}_0$) as the *base topology*, while a metric-induced topology with $c > 0$ can result in more extensive condensation, in which arbitrary connected subgraphs of internal nodes may collapse into a single node. The following example illustrates this behavior for the specific case of loss topologies.

Consider the physical tree topology shown in Figure 1(a). Node 0 is the server, nodes 4, 5, 8, 11, 12, and 14 are the clients and the remaining nodes are routers. The links in Figure 1(a) are labeled with the actual loss rates on these links. Figure 1(b) shows the loss topology $\mathcal{T}_0$ for this physical tree when $c = 0$. Notice that in $\mathcal{T}_0$, paths with intermediate nodes of unit out-degree (e.g. the path between nodes 0 and 3 and the path between nodes 6 and 8) are collapsed into a single (logical) link. Figures 1(c) and 1(d) show the loss topologies for this physical tree when $0.03 < c \leq 0.04$ and when $0.04 < c \leq 0.05$, respectively.

[3] Note that the values on adjusted links must be updated in a metric-specific way after each collapse operation. We discuss the significance of this in Section 3.2.



(a) Physical tree (b) Loss tree ($c = 0$) (c) Loss tree ($0.03 < c \leq 0.04$) (d) Loss tree ($0.04 < c \leq 0.05$)

Figure 1: Relationship between physical and loss topologies under various sensitivity parameter (c).

These topologies are obtained from T_0 by collapsing internal links with loss probabilities that are less than the sensitivity parameter c. For the sake of simplicity in this example, we have taken the liberty of merging loss probabilities on consecutive links by using simple addition.

3.2 MINT Inference and Labeling from End-to-End Measurements

The topology inference framework we present next applies to a broad class of metrics (encompassing packet loss rates, propagation delay, and available network bandwidth) satisfying two technical conditions. We will define those conditions next, then go on to demonstrate how our inference techniques can be (1) recursively applied to large topologies, (2) applied to incorporate observations taken at multiple points in time, and (3) applied to incorporate observations taken at multiple points in space.

To define these conditions, we need a bit more terminology. We refer to path p_i as a *subpath* of p_j if the sequence of edges forming p_i is a subsequence of the edges forming p_j. Let us then denote $p_{i.j}$ as the maximal subpath common to both p_i and p_j, i.e. $p_{i.j}$ is the shared portion of p_i and p_j. From these definitions, when p_i is a subpath of p_j, $p_i = p_{i.j}$. A metric must satisfy the following two properties to be amenable to our technique:

Monotonicity: A metric f is *monotone* if for any pair of paths p_i, p_j for which p_i is a subpath of p_j, $f(p_i) \leq f(p_j)$.

Separability: A metric f is *separable* if for any path p composed of the union of two disjoint subpaths p_i and p_j, $f(p_j) = g(f(p), f(p_i))$ for some function g.

Note that both of the above properties hold naturally with respect to the metrics of packet loss rates and propagation delay. Using the example of propagation delay, this metric is clearly monotone, as delay along a subpath is strictly smaller than that along the full path, and simple subtraction can be used as the separability function g. Available bandwidth is another example of a metric which fits the definitions above, with the caveat that the definition of monotonicity be extended to include monotonically *non-increasing* functions, by reversing the inequality in the definition.

We are now ready to show how to infer and label topologies induced by a metric f using end-to-end measurements.

Theorem 1 *Given a procedure that enables the evaluation of $f(p_a)$, $f(p_b)$, and $f(p_{a.b})$ for some monotone metric f between a source s_0 and any two endpoints a and b, one can efficiently* infer *the base topology induced by f over an arbitrary physical topology $\mathcal{T}$. In the event that f is separable, one can also efficiently* label *the topology induced by f for any sensitivity parameter $c > 0$.*

Proof: (Sketch). We first demonstrate how to recursively infer the base topology based on end-to-end evaluations of a monotone function f. Consider a set of n endpoints $s_1, s_2, \ldots s_n$. Sort all pairs of endpoints (s_u, s_v) sorting on the value of $f(p_{s_u.s_v})$. Let s_i and s_j be the pair[4] of endpoints for which $f(p_{s_i.s_j})$ is maximum.

Reduction: Monotonicity ensures that path $p_{s_i.s_j}$ cannot be a subpath of any other path connecting the source s_0 to any endpoint other than s_i and s_j. Thus there exists an internal node r at which the paths from s_0 to s_i and s_j diverge; moreover, r is *not* on any path connecting s_0 to any endpoint other than i and j. The subtree rooted at r is therefore completely specified.

Recursion: The above construction identifies an internal node r in the logical topology, and gives a method for computing $f(p_r) = f(p_{s_i.s_j})$. This construction therefore allows us to remove s_i and s_j and replace them with r, thereby reducing the problem of finding the logical topology connecting a server s_0 to n endpoints to the smaller problem of finding the logical topology connecting that same server to strictly fewer than n endpoints. Applying this reduction recursively, one can produce the base topology with respect to f.

In the event that the metric f is also separable, we can prove the second part of Theorem 1 for any sensitivity parameter $c > 0$ by repeatedly applying the following step:

Compression and Relabeling: In a bottom-up fashion, we collapse an *internal* node r_i into its parent node r_j iff $g(f(p_{r_i}), f(p_{r_j})) < c$. Recall that collapsing r_i into r_j entails deleting edge (r_i, r_j) from the tree and attaching all children of r_i to r_j. The label of the edges connecting former children of r_i to r_j must be updated; by the separability condition, that is by an amount equal to $g(f(p_{r_i}), f(p_{r_j}))$. The result is a labeled topology and we refer the reader again to Figure 1 for a detailed example. ■

3.3 Integrating MINT Snapshots

Due to the non-stationarity of the metrics that one might hope to observe with a set of end-to-end observations over time, it is evident that the labels placed on the edges of a metric-induced topology will change over time. Moreover, at different points in time, measured observations over a link or set of links may fail to reach the threshold specified by the sensitivity parameter. For this latter reason, time-varying observations of a metric-induced topology measured from the same endpoints may contain different sets of observable internal edges. Therefore, one can hope to integrate a set of MINT snapshots generated at different points in time to produce a topology which includes the *union* of all internal edges observed in the snapshots It is natural to wonder whether one can produce such an integrated topology from a set of mutually consistent topological snapshots efficiently. The following theorems demonstrate that this can in fact be achieved.

Definition 4 *Consider a set of unlabeled topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ rooted at the same source and spanning destination sets $E_1, E_2, \ldots E_k$ respectively. We say that these topologies are* mutually consistent *if there exists a tree $\mathcal{T}'$ spanning all destinations in $\bigcup E_i$ from which we can generate $\mathcal{T}_i$ for any i by repeated application of the collapse operation defined earlier.*

Theorem 2 *The minimal tree T' spanning a set of mutually consistent topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ connecting a server to a set of m clients is unique and can be constructed in $O(n.m^2.log(d))$ time.*

[4] In general, there may be a set of nodes whose pairwise evaluations are all equal and maximum. These nodes would all be grouped together in the reduction operation.

Proof: The algorithm to produce T', a proof of its correctness, and bounds on its running time are provided in the appendix. ■

Another important problem that is similar to the integration of time-varying observations is that of integrating observations made from *spatially distinct* vantage points. As motivated in the introduction, a content delivery network (CDN) consisting of a number of distributed servers already performs a vast number of end-to-end observations in the normal course of daily operations and could benefit from a methodology to integrate these techniques. We provide the following additional definitions to generalize the notions above to enable the integration of metric-induced topological snapshots made from different points in space—namely, how to produce a graph which is mutually consistent with a set of consistent snapshots collected from a set of distributed servers.

Definition 5 *Consider a set of metric-induced topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ rooted at* different *sources. We say that these topologies are* mutually consistent *if there exists a graph G spanning all endpoints in $\bigcup \mathcal{T}_i$ and when G is restricted to the routing tree induced by IP routing and spanning endpoints in $\mathcal{T}_i$, $G|\mathcal{T}_i$, $\mathcal{T}_i$ can be generated from that subgraph by repeated application of the collapse operation.*

Symmetry: A metric f is *symmetric* iff for any path p_i the value $f(p_i)$ is equal as observed from either end of p_i. Many metrics satisfy the symmetry property. These include Capacity Bandwidth, propagation delay, transmission delay, ...

Theorem 3 *Given a set of n sources s_0, s_1, ..., s_{n-1} and a procedure that enables the evaluation of $f(p_{s_i})$, $f(p_{s_j})$, and $f(p_{s_i,s_j})$ for some monotone, separable and symmetric metric f between any source s_k and any two other sources s_i and s_j, one can efficiently* infer *and* label *the base topology $\mathcal{G}$ induced by f over the physical topology induced through IP routing and connecting the n sources for any sensitivity parameter $c > 0$.*

Proof: Refer to the appendix for the proof. ■

As with temporally distinct observations, we have an efficient algorithm for determining whether a set of topologies are mutually consistent, and if so, generating a spanning graph G as in the definition above. With these two combining mechanisms, we may merge a set of metric-topological views collected at multiple points in time and multiple points in space to produce an (unlabeled) topology incorporating the available information. Again, due to non-stationarity of observed values over time (more discussion of this in our experimental results section), we do not propose a mechanism to accurately *label* a topology integrated from multiple views in space-time.

4 Loss Topology Identification Using Unicast Probing

As established by Theorem 1, any end-to-end procedure which is capable of labeling a metric-induced topology with two clients can be recursively applied to label an arbitrarily large metric topology provided the metric is monotone and separable. However, since the exact nature of a particular procedure used to label a topology with respect to a particular metric may vary widely, the extent to which such a labeling process scales and the accuracy of the procedure must be analyzed in the context of the particular algorithm used. In this section (and for the remainder of this paper), we focus our presentation on applying the results

presented earlier with respect to a particular metric—namely *loss rate* under a specific procedure, which we describe now.

Overview of Bayesian Probing: We describe our approach to the loss topology inference and labeling using the Bayesian Probing (BP) technique in the spirit of work proposed in [21]. Consider clients 11 and 14 in the topology shown in Figure 1(a). Using the terminology of Section 3, paths p_{11} and p_{14} from the server to each of these clients can be partitioned into two subpaths: the portion that is *shared* between the two paths, $p_{11.14}$, and a portion that is unique for each path. Specifically, L_6L_9 is a shared segment, whereas $L_{10}L_{11}$ and $L_{13}L_{14}$ are not. The BP technique provides us with a simple probing methodology that enables the estimation of p_i, p_j and $p_{i.j}$ for all i, j as required by Theorem 1. To that end the technique uses two types of probe sequences:

Definition 6 *A* 1-packet probe sequence $S_i(\Delta)$ *is a sequence of packets destined to client* i *such that any two packets in* $S_i(\Delta)$ *are separated by at least* Δ *time units.*

Definition 7 *A* 2-packet probe sequence $S_{i,j}(\Delta, \epsilon)$ *is a sequence of packet-pairs where one packet in each packet-pair is destined to* i *and the other is destined to* j*, and where the intra-pair packet spacing is at most* ϵ *and the inter-pair spacing is at least* Δ *time units.*

1-packet probe sequences provide a baseline loss rate over end-to-end paths while 2-packet probe sequences enable measurement of loss rates over shared links. The intuition is that because of their temporal proximity, packets within a packet pair have a high probability of experiencing a shared fate on the shared links. If the incidence of shared loss on the shared links is high, this leads to an increased probability of witnessing coupled losses within a packet pair. While we describe appropriate settings of Δ and ϵ in the experimental results, we generally find that setting ϵ to be on the order of a millisecond and Δ to be on the order of hundreds of milliseconds achieves high dependence and ensures independence, respectively. Using statistics of successful packet delivery of 1-packet and 2-packet probes, the BP techniques enables the estimation of the magnitude of packet losses on the shared segment of paths from a server to two clients.

Basic BP Assumptions: As spelled out by the authors of [21], the BP probing technique and associated analyses (leading to the estimation of packet loss rates) are subject to several significant assumptions, which we enumerate below.

1. Link loss rates are stationary over the time scale it takes the BP technique to converge.
2. Losses on the links occur only due to queue overflows.
3. $\forall i, j$: Losses on link L_i are independent from losses on link L_j.
4. There exists a reliable feedback mechanism to determine the fate of a given probe packet.
5. The temporal constraints imposed on probe sequences are preserved throughout the journey of the probes from sender to receivers.

While the above assumptions can be achieved in simulation, and indeed are satisfied in our simulation studies presented in Section 5 (with the exception of assumption 5), *none* could be guaranteed for our experiments over the Internet presented in Section 6! Nevertheless, our results show that the BP approach is robust enough to enable accurate Internet loss topology inference and labeling. We proceed to describe our experimental findings in the remainder of the paper.

5 Performance Evaluation

In this section we present results of extensive `ns` simulations that demonstrate the accuracy, convergence and robustness of our approach. In the subsequent and final section of this paper, we report on our implementation and experiments we have conducted in the wide-area.

5.1 Simulation Environment

Bayesian Probing Technique: The BP technique requires specification of the Δ and ϵ parameters describing the temporal constraints imposed on 1-packet and 2-packet probe sequences. In the experiments we present, each probe sequence was generated using an independent Poisson process with a mean probing rate of 5 probes/sec, or 200ms average inter-packet spacing. A lower bound on Δ was not guaranteed. For 2-packet probing processes, the value of ϵ was set to 0; that is packets within a packet-pair were sent back-to-back, with no time separation. The 2-packet probes in a probe sequence alternate between the two possible packet orderings.

Link Baseline Model: Each of the links in our simulations is modeled by a DropTail queue. The link delays were all set to 40ms and the link buffer sizes were all set to 20 packets. Each link was subjected to background traffic resulting from a set of Pareto ON/OFF UDP sources with a constant bit rate of 36Kbps during the ON times with a packet size of 200 bytes. The average ON and OFF times were set to 2 and 1 seconds, respectively. The Pareto shape parameter (α) was set to 1.2. After a "warm-up" period of 10 seconds, the probing processes (and associated inference/labeling processes) begin.

To represent the various levels of congestion that any of these links may exhibit, we have chosen three sets of parameters that result in "High", "Mild", and "Low" levels of congestion. The baseline parameter settings for these congestion levels (and the resulting loss rates) are tabulated in Table 2.

Parameter Setting	Congestion Level		
	High	Mild	Low
Link Bandwidth	1Mbps	1Mbps	100Mbps
# of background flows	60	56	44
Observed Loss Rate	7-15%	$< 7\%$	$< 1\%$

Table 2: Settings used (and resulting loss rates) for the three levels of congestion considered

Our choice of very high loss rates (7-15%) for highly congested links was meant to stress-test our technique under severe congestion (we observed many instances of lightly congested links in the wide area when testing our implementation). We set the value of the sensitivity parameter c to a fixed loss rate of 0.04. This value was chosen empirically based on our experimental set-up; we imagine that in general, the sensitivity parameter will have to be chosen in an application- and metric-specific way. We note that the exact upper bound on link losses is unimportant since the sensitivity parameter setting (c=4%) is small enough to observe links with losses $> 4\%$. Our simulation results are actually slightly better under lower maximum loss rates because the presence of highly lossy links slows BP's ability to characterize subtopologies downstream of those links.

Topology Baseline Model: In order to test our solutions of the inference, labeling, and augmentation problems in a controlled environment, we generated a baseline test set of random tree topologies of varying shape, depth, and with variable fanout (up to degree 4) on 14 nodes, of which 5 leaves were then selected as clients. Details of our methodology are provided in the full version of the paper. The congestion level for each tree edge was then chosen at random from the link baseline models with the following distribution:

50% Low, 30% Mild and 20% High.

In order to enrich the topology between the server and these 5 clients, we have added 8 more "dummy" clients. These 13 (base + dummy) clients represented the leaf nodes of the tree. A bottom-up iterative process was used to generate the internal nodes of the tree, first by picking a degree d at random from a distribution $\mathcal{D}$, then selecting d leaves or existing internal nodes to connect to this new parent node. This process of adding internal parent nodes continued until only a single root node remained. Once this process terminated, the dummy clients and stub links were removed, leaving multi-hop routes to the base clients. Clearly, the distribution $\mathcal{D}$ is a significant determinant of the trees we construct. We used a distribution in which d was 2 with probability 0.65, d was 3 with probability 0.3 and 4 with the remaining probability (0.05).

5.2 Performance Metrics

We now introduce the three metrics used to evaluate our unicast-based loss topology inference and labeling techniques—namely, *inference accuracy*, *inference discrepancy*, and *labeling error*. The first two metrics are used to evaluate the goodness of a loss topology inference technique, whereas the third is used to evaluate the goodness of a loss topology labeling technique. In practice, all three of these metrics will be computed by running experiments over a representative set of similar trees, computing the metrics over those inputs, then averaging the results to derive an estimate.

In each of the definitions below, we assume that the inference/labeling process starts at time $t = 0$, that $1 \leq k \leq N$ refers to the experiment under consideration, that $0 < i, j \leq M$ refer to clients (or endpoints), and that $0 < l \leq L$ refers to a link of a given loss topology (tree).

Definition 8 *The inference Accuracy $A(t)$ of a loss topology inference strategy at time t is defined as the probability that the strategy yields the correct loss topology at time t.*

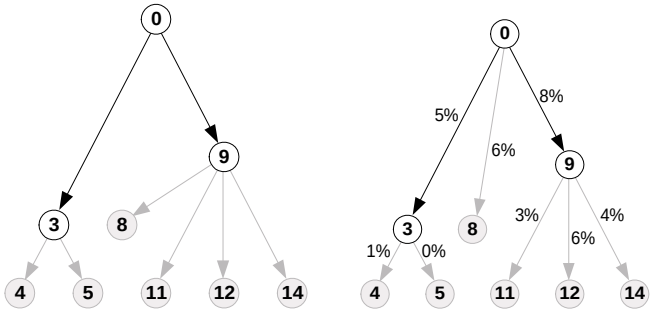
In our experiments, to measure accuracy at time t, we calculate the percentage of the simulation experiments in which the correct loss topology was identified at time t. The accuracy metric is an absolute metric, in the sense that it does not allow for a quantification of how "close" an inferred loss topology is to the exact loss topology in the event that it is inaccurate. The discrepancy metric provides such a quantification.

Definition 9 *The inference Discrepancy $D(t)$ of a loss topology inference strategy on a tree T at time t is given by:*

$$D(t) = \sqrt{\frac{\sum_{i,j:i\neq j}(\hat{d}_{i,j}(t) - d_{i,j})^2}{\binom{M}{2}}}$$

where $d_{i,j}$ denotes the depth of the least common ancestor of a pair of clients i and j in the correct loss topology induced by T and $\hat{d}_{i,j}(t)$ denotes the depth of least common ancestor of a pair of clients i and j in the inferred loss topology at time t.

To give an intuition for the discrepancy metric, consider the topology shown in Figure 1(a) and assume that as a result of applying a topology inference procedure with a $c = 0.05$, the topology shown in Figure 2(left) is obtained. The inferred topology is not identical to the correct $\mathcal{T}_{0.05}$ loss topology shown in Figure 1(d) and the *discrepancy* between the inferred topology and $\mathcal{T}_{0.05}$ is $\sqrt{\frac{3}{15}} = 0.447$.



Inferred Tree (a) Labeled Tree (b)

Figure 2: Illustration of the use of the Inference Discrepancy and Labeling Error metrics.

Definition 10 *The labeling Error $E(t)$ of a loss topology labeling process on tree T at time t is:*

$$E(t) = \sqrt{\frac{\sum_{l=1}^{L}(\hat{e}_l(t) - e_l)^2}{L}}$$

where e_l denotes the correct loss probability (i.e. label) for link l and $\hat{e}_l(t)$ denotes the measured loss probability for link l at time t.

To give an intuition for the labeling error metric, consider the topology shown in Figure 1(a) and assume that as a result of applying a topology labeling procedure with a $c = 0.05$, the labeled topology shown in Figure 2(right) is obtained. Obviously, the labels on that topology are not identical to the labels on the $\mathcal{T}_{0.05}$ loss topology shown in Figure 1(d). The *labeling error* can be calculated to be $\sqrt{\frac{0.02^2+0.01^2+0.01^2}{8}} = 0.00866$. This quantity can be viewed as the average deviation in the labeling of an arbitrary link.

5.3 Inference of Loss Topology Experiments

Experimental Setup: In order to determine the accuracy of our topology inference technique, we generated 20 5-client baseline trees at random (as described earlier). Our objective is to infer the loss topology between a server and the 5 clients in each one of the baseline trees. We ran the loss topology inference technique from the server (root node) by creating an `ns` agent that sends the probes, collects statistics about these probes, calculates the needed estimates, and executes our topology inference procedure. For each one of the 20 randomly generated trees, we ran this experiment 20 times; each time seeding the cross-traffic with a different random seed. The results were then averaged over these 400 experiments.

Results: Figure 3 shows the accuracy and discrepancy metrics for our loss topology inference experiments as functions of time for different values of the sensitivity parameter c. Our inference technique converges rapidly as a function of the number of probing rounds. Figure 3 indicates that both the accuracy and discrepancy metric settle to within 10% of their steady-state values within 50 seconds.[5]

Figure 4 shows that both the accuracy and discrepancy metrics improve as the value of the sensitivity parameter c increases. We quantified this dependence by measuring both accuracy and discrepancy at

[5]Since the probing rate was set to 5 probes/sec, it follows that the accuracy and discrepancy metric settle to within 10% of their steady-state values within 250 probing rounds.

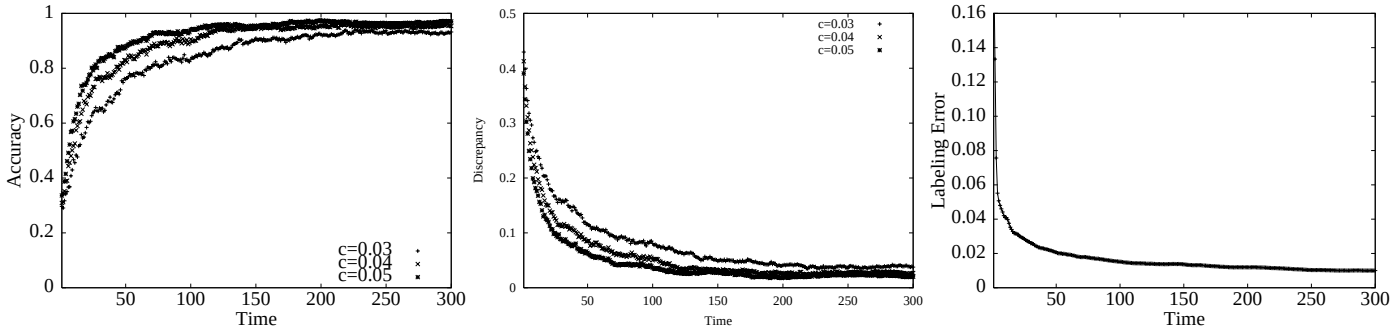


Figure 3: Accuracy (left), Discrepancy (middle), and Labeling Error (right) of Loss Topology over Time

three different points in time for various values of c. Figure 4 shows that there is indeed an "inflection point", after which both accuracy and discrepancy start deteriorating. Notice that the inflection point is dependent upon our choice for what constitutes "High", "Mild", and "Low" levels of congestions (see Table 2); a well-chosen value of c allows a correct disambiguation between what an application considers significant losses versus insignificant losses.

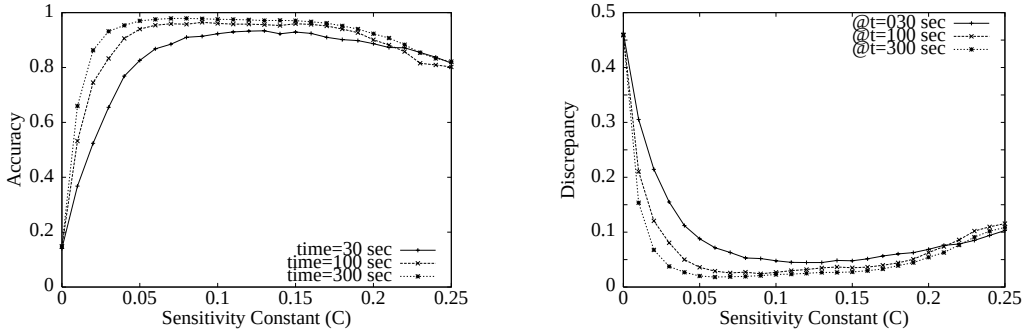


Figure 4: Effect of Sensitivity Parameter on Accuracy (left) and Discrepancy (right)

5.4 Labeling of Loss Topology Experiments

Experimental Setup: The setup of the labeling experiment was the same as the loss topology inference experiment; except that the server agent is furnished the correct loss topology and runs the labeling algorithm to label this topology. The results are still averaged over 400 experiment runs for 20 randomly-generated trees.

Results: Figure 3 (right) shows the labeling error as a function of time. The labeling error converges to within 1% of the actual links loss rates. We noted that in most of the cases the labeling results are very close to the actual losses; except for the cases where the shared links between 2 clients contain more than one highly congested bottleneck. In this case, cross-traffic packets intervene between the packet-pair probes and space the packet-pair out after the first bottleneck, causing less correlated behavior when passing through the second bottleneck. This leads to an accurate assessment of the first bottleneck loss rate while most of the second shared bottleneck loss rate is assigned incorrectly to the independent links.

5.5 Simulation of Large-Scale Topologies

The last observation in the preceding section points to one of several issues which may negatively impact

the performance of the BP approach to loss topology inference. While the server can ensure that packets used in 2-packet probes satisfy the requirement that emission of the two packets in a 2-packet probe be separated by no more than ϵ, it cannot "guarantee" that such a constraint is satisfied throughout the network. In practice, it is likely that the separation between packets in a packet-pair increases as the number of hops traversed increases.

To assess the robustness of our approach and its effectiveness for larger loss topologies, we have conducted experiments on a relatively large tree. We used the same tree generation procedure described in Section 5, except that the number of base clients was increased to 50 and the number of dummy clients was increased to 200. The resulting trees had an average depth of over 8 levels and in excess of 400 nodes. Also, the congestion level for each of the links of the tree was selected from one of the three link baseline models described earlier. We used a distribution with 90% Low, 7% Mild and 3% High congestion to keep the end-to-end congestion level reasonable. As before, we ran 20 inference and labeling experiments on that tree. Figure 5 (left) shows the inference accuracy for this large-scale simulation. While the convergence of loss topology inference for this large tree is slightly slower than that presented in Figure 3, the accuracy of the inference in steady-state remains robust (over 90% after 150 seconds). The results for the discrepancy and labeling error metrics were equally robust as shown in Figure 5 (middle) and 5 (right), respectively.

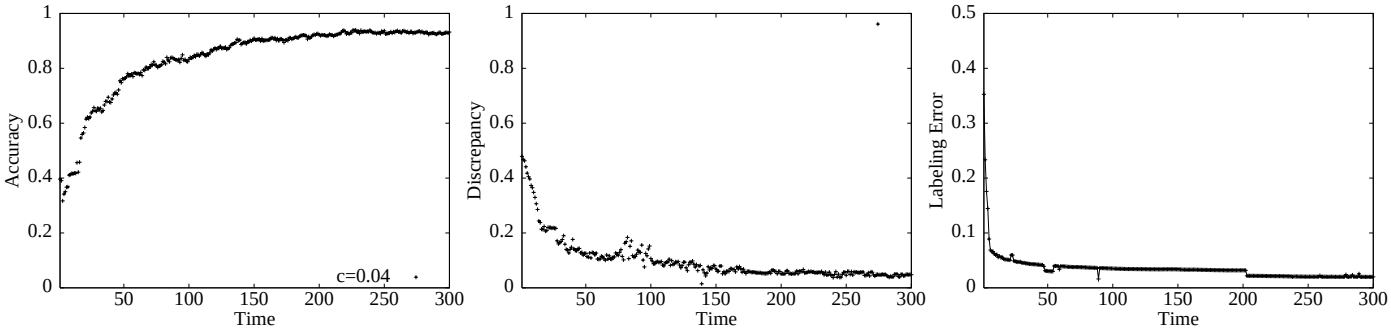


Figure 5: Inference accuracy (left), discrepancy (middle), and labeling error (right) over time for a tree with 50 leaves and 200 internal nodes.

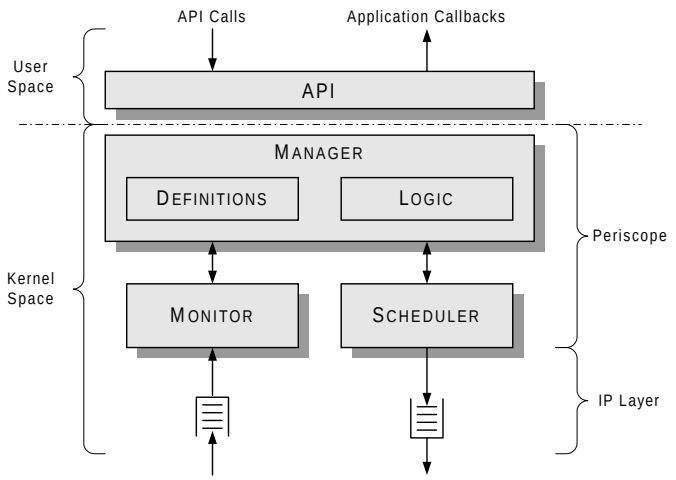
6 Implementation and Deployment

We have developed a Linux API called Periscope (a Probing Engine for the Recovery of Internet Subgraphs) that implements the functionality necessary to infer and label metric-induced topologies. Using that API, we have developed a tool to infer and label loss topologies from a server to a set of clients. In this section, we briefly describe the architecture of Periscope and report on results we obtained using Periscope to infer, label, and characterize loss topologies between a server and a set of clients across the Internet.[6]

6.1 Periscope: Architecture

Periscope server-side functionality consists of: (1) orchestrating probe transmission, (2) maintaining probe loss statistics, and (3) running the inference and labeling processes. Periscope requires *no* support from clients beyond the ability to respond to ICMP ECHO REQUESTs. Figure 6 (left) depicts the main components of the Periscope architecture.

[6]The Periscope API and the loss topology inference and labeling tool will be made public (via FTP) once the paper is selected for publication.




```
typedef struct{
  int max_phase;
  int probes_per_sec;
  int results_reporting_frequency;
  int probe_size;
} param_t;

typedef struct{
  struct timeval send_time;
  int group_id;
  int flow1_id;
  int flow2_id;
  int which_flow;
} payload_t;

int PS_open(void);
int PS_new_group(int ctrl_fd,param_t params);
int PS_register(int ctrl_fd,int gid,u_char *host);
int PS_activate_group(int ctrl_fd,int gid);
```

Figure 6: PERISCOPE: Architecture (left) and API (right).

The `Manager` keeps a record of all endpoints (i.e. clients) under consideration. Endpoints are partitioned into application-defined *groups*—groups of clients specified by applications through an API call. The inference and labeling procedures are applied on flows to endpoints belonging to a single group. Periodically, the `Manager` reports inference and labeling results through application callbacks.

The `Scheduler` uses a timer for each group. Whenever a timer associated with a group of endpoints expires, a new probe for this group is inserted in the IP stack for transmission. This probe consists of a packet-pair destined to two of the group's endpoints. The `Scheduler` ensures that the packets within a packet-pair are inserted back-to-back on the top of the IP stack.[7] This provides the correlation needed for BP probing. The endpoints to which probes are targeted are selected by the `Scheduler` in a random fashion to avoid synchronization effects (see below).

The `Monitor` keeps track of the loss statistics for each endpoint in each group. These statistics are updated as a result of the receipt of an ICMP ECHO REPLY from an endpoint.

PERISCOPE is implemented in the kernel. By implementing the scheduling and monitoring functionalities in the kernel, PERISCOPE minimizes User/Kernel boundary crossings. The User/Kernel boundary is crossed only during group setup, flow registration and during periodic application callbacks to report inference/labeling results. This optimization is valuable for busy servers.

The interface between applications and PERISCOPE is done through the use of control sockets. System calls (discussed later) are translated through `ioctl` calls to perform appropriate actions in the kernel. An application uses the `select()` system call to receive PERISCOPE callbacks. This approach (control socket + select + ioctl) restricts code changes caused by PERISCOPE to the networking stack and provides a well-defined and flexible interface for applications.[8]

6.2 PERISCOPE: API

The PERISCOPE API frees the application from having to manage the probing, statistics collection and

[7] In order to send two packets back-to-back we extended the kernel library with a function that sends two packets back-to-back with a single system call.

[8] Our approach (control socket + select + ioctl) was influenced by the architecture of the Congestion Manager [2], with the possibility of future integration in mind.

inference processes. Using the PERISCOPE API, an application can determine the loss topology between itself and a set of clients. This is done by having the application: (1) create a new *group* and provide the parameters to be used by PERISCOPE (e.g., probing rate, ECHO packet size, frequency of callbacks, sensitivity parameter to be used for inference, etc.), (2) register the endpoints to be considered as part of this group, (3) activate the group, and (4) wait for feedback.

Figure 6 (right) shows the PERISCOPE API (data structures and system calls). The structure *param_t* contains the different parameters that define the settings for a specific group. A *phase* is defined as a sequence of packet-pair transmissions that cover all possible ordered combinations of endpoints in a group. Packet-pairs in a phase are sent in a random permutation to avoid possible synchronization effects. The *max_phase* field of the *param_t* structure defines the total number of phases that the application wishes PERISCOPE to perform. The *probes_per_sec* field defines inter-probe times. The *results_reporting_frequency* defines the period of PERISCOPE reports in terms of number of phases. A PERISCOPE header is appended as a payload to ICMP ECHO REQUEST packets, the size of which is defined by the *probe_size* field.

`PS_open()` creates a new socket of type SOCK_PS, a PERISCOPE-defined type used for the transmission and receipt of probes and of protocol type IPPROTO_ICMP. `PS_new_group()` allocates a new group under a previously opened *ctrl_fd* socket file descriptor. The *parameters* of this group are passed along as arguments and the return value from this function is a group identifier. `PS_register()` registers a new endpoint with group *gid* associated with the *ctrl_fd* socket. Finally, `PS_activate_group()` activates *gid*'s probing timer, thus starting the periodic probe transmission and statistics collection procedures for the group.

6.3 PERISCOPE: Validation

To validate the ability of PERISCOPE to correctly infer and label loss topologies in an Internet setting, we hand-picked a set of seven hosts and used PERISCOPE to infer and label the loss topology to these endpoints from a local server (Pentium II processor running RedHat Linux version 2.2.14). The seven endpoints were selected to ensure the existence of different lossy paths that are shared between the server and various subsets of endpoints. In addition, by placing the server below a slow uplink, we ensured the existence of a (possibly) lossy path between the server and all endpoints. These choices were all made with the goal of stress-testing our inference and labeling techniques in mind.[9] Figure 7 depicts the logical topology between the server and the seven hosts, constructed by collapsing chains of hops in the tree as explained in Section 3.1 and Figure 1). Intermediate router IP addresses were obtained through the use of *traceroute*. The server is in the continental U.S. Hosts A,B and C are in China with hosts A and B on the same LAN of Beijing University of Aeronautics and Astronautics and host C in Northeast China Institute of Electric Power Engineering. Hosts D and E are in Egypt, on the same LAN of the Arab Academy for Science and Technology (AAST). Hosts F and G are in Italy at two different universities: Politecnico di Bari and Universita Degli-Studi di Bergano.

To validate the accuracy of PERISCOPE we need to establish a "reference" against which we could compare the inferred and labeled loss trees we obtain for a given sensitivity parameter c. The logical tree (shown in Figure 7) is such a reference for $c = 0$. Obtaining such a reference for a non-zero sensitivity parameter is impossible since it requires knowledge of loss rates on all links of the logical tree. Moreover,

[9] Validating our tool requires observing loss-topologies of appreciable structure—hence our choice of an inter-continental set of endpoints. Internet loss topology characterizations (see discussion in Section 7) to small sets of random endpoints (from CAIDA/NLANR logs) rarely yielded rich/interesting structures. The depicted topology is meant to be illustrative, not representative. Also, experiments to the selected set of endpoints didn't consistently reveal high losses. Figure 7(right) was constructed only after integrating consistent inferred loss topologies viewed at different times.

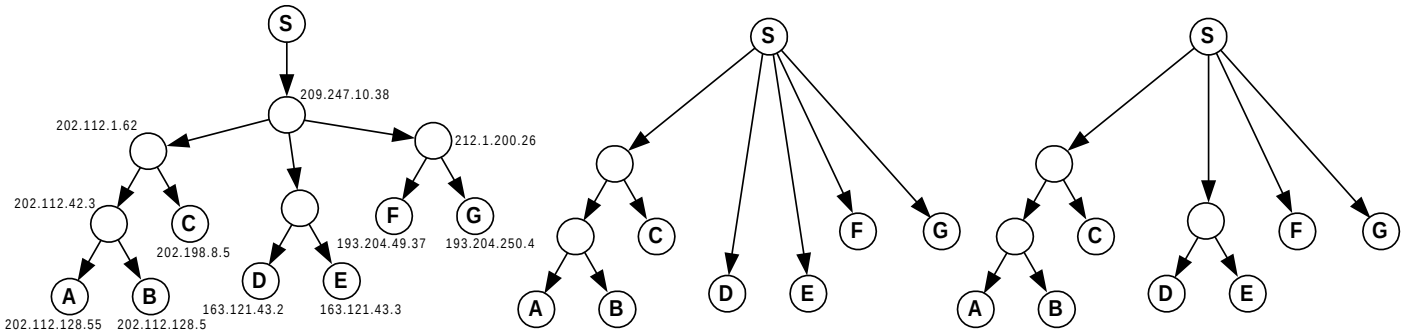


Figure 7: PERISCOPE Validation: Logical tree used as a test case (left), most frequent inferred loss tree (middle) and minimal loss tree spanning all inferred trees (right).

loss rates cannot be assumed stationary for the duration of a PERISCOPE experiment and may not always be above the sensitivity parameter specified in PERISCOPE.

While the logical tree in Figure 7 cannot be used to directly validate loss trees inferred by PERISCOPE it can be used to check whether the loss trees generated by PERISCOPE are *mutually consistent*, as defined in Section 3. We performed 20 experiments using PERISCOPE to infer and label the loss topology to the seven endpoints of the logical topology in Figure 7. These 20 experiments were conducted at different times. Each experiment consisted of 100 probing phases with 64-byte probes. At a probing rate of 5 probes/sec, it takes PERISCOPE about 4 minutes to complete 100 phases of probing. Notice that this time could be decreased by reducing the number of phases or by increasing the probing rate. Indeed, in most experiments, the loss topology tree "stabilized" within less than 10 phases—i.e. less than 24 seconds. However, increasing the probing rate is not desirable because it may result in the violation of the inter-probe independence assumption of the BP approach alluded to in Section 4.

Figure 8 (left) shows the percentage of PERISCOPE inferred trees that are found to be *inconsistent* with the logical tree in Figure 7 for various values of the sensitivity parameter c. This relationship is shown for three different periods of running PERISCOPE—namely, after 20, 40 and 80 phases. As expected, the inconsistency of the inferred tree decreases as the sensitivity parameter increases.

As explained in Section 3, the non-stationarity of losses on the various links in a logical topology makes it unlikely that all of the potentially lossy links will be observable in a given experiment at a given time. Thus, one would expect that the loss topologies inferred by PERISCOPE will be different when run on the tree in Figure 7 (left). Indeed, PERISCOPE inferred six different loss topologies. Over the 20 experiments we conducted, the most frequently inferred loss topology tree is shown in Figure 7 (middle). This tree was inferred 11 times at times ranging between 3am and 7am EST (consistent with the fact that the lossy paths were the ones connecting our server to the hosts in China).

Using the procedure described in Theorem 2, we constructed the minimal loss tree spanning all six of the loss topologies inferred by PERISCOPE when $c = 0.01$. The resulting tree (which itself is not one of the trees inferred by PERISCOPE) is shown in Figure 7 (right). Clearly, that tree is consistent with the logical tree depicted in Figure 7 (left).

To validate the labeling accuracy of PERISCOPE, we implemented a simple tool which repeatedly probes *all* nodes, including internal nodes (i.e. along subpaths to endpoints), of the logical tree of Figure 7 *concurrently* with our use of PERISCOPE.

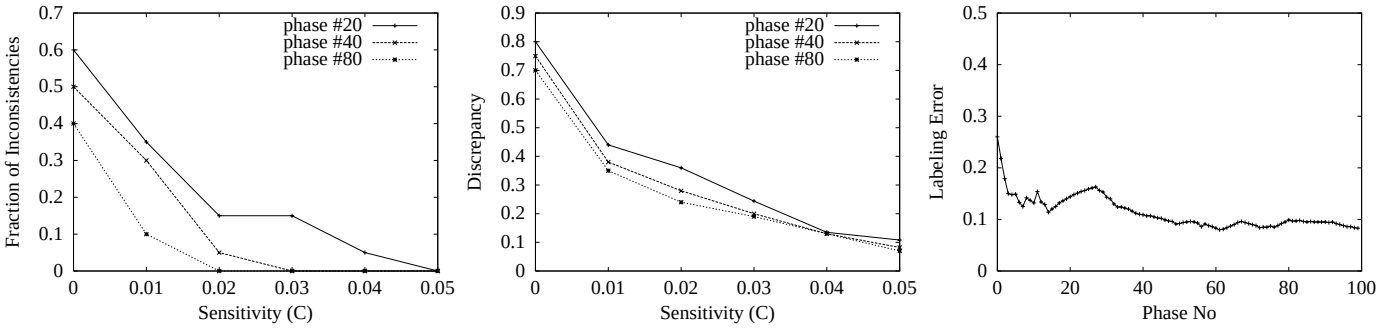


Figure 8: PERISCOPE: Inference inconsistency (left) discrepancy (middle) and labeling error (right) for the Internet topology shown in Figure 7

Probing is done by sending ICMP ECHO REQUEST packets and for each node the percentage of probes for which we do not receive ICMP ECHO REPLY packets is recorded[[K]].

Loss statistics obtained from this Poisson probing tool are compiled to yield the loss rates on the various links of Figure 7. Finally, the resulting labeled tree is compressed using the same sensitivity parameter of PERISCOPE. The discrepancy and labeling error between the loss trees obtained using this tool and those obtained using PERISCOPE are measured and presented in Figure 8.

Figure 8 (middle) shows the discrepancy between the PERISCOPE-inferred loss trees and the loss trees obtained using the tool described above for various values of c. The results show that the discrepancy decreases slightly as c decreases. To demonstrate the convergence characteristics of PERISCOPE, Figure 8 (right) shows the reduction in the labeling error as the number of phases executed increases from 0 to 100 phases, spanning approximately 4 minutes.

7 Conclusion

One of the defining principles of the network protocols used in the Internet lies in their ability to manage and share network resources fairly across competing connections. This is a notable engineering achievement, especially in light of the fact that individual connections exert distributed control over their transmission rates. But the fine-grained autonomy that connections exert coupled with our limited understanding of the interactions that multiple connections impose limits the degree to which network resources can be tightly controlled. In our ongoing work we investigate circumstances in which better diagnosis of network resources can be obtained, which we hope will lead to improved control mechanisms.

Summary: In this paper, we propose the use of metric-induced topologies as abstractions that enable a compact representation of the shared congested network resources that need to be managed by Mass servers. We instantiate our approach using a specific metric of interest—namely, packet loss rates. To that end, we present an analytical framework for the multiscale characterization of Internet *loss topologies*, and we show how end-to-end unicast packet probing techniques could be used to (1) infer a loss topology and (2) identify the loss rates of links in an existing loss topology. We reported on simulation, implementation, and Internet deployment results that show the effectiveness of our approach and its robustness in terms of its accuracy and convergence over a wide range of network conditions.

Ongoing Work: The Internet experimentation results presented in this paper were aimed at validating the effectiveness of PERISCOPE in identifying loss topologies between a server and a set of WAN endpoints.

We are currently conducting a number of experiments (using PERISCOPE) that we hope will shed light on the characteristics of Internet loss topologies. For example, using PERISCOPE, we characterize the *depth* of Internet loss trees that are observable from a single vantage point. The depth of a loss tree is a characterization of the maximum number of successive "bottlenecks" shared with other clients from the same server, where a bottleneck is defined as a path with a loss rate that is larger than the sensitivity parameter.[10] Early results suggest that for small number of clients, the likelihood of shared losses—while small—are not insignificant.

There has been substantial recent work on discerning one or more network-internal features of interest. Our work does so using only end-to-end observations, and with unicast messaging. Ideally, we would like to accomplish topology identification passively (i.e., using feedback from established TCP connections), and this is a central goal of our future work.

[10] Obviously, the depth of a loss tree depends on the number of end-points as well as the sensitivity constant for that loss tree.

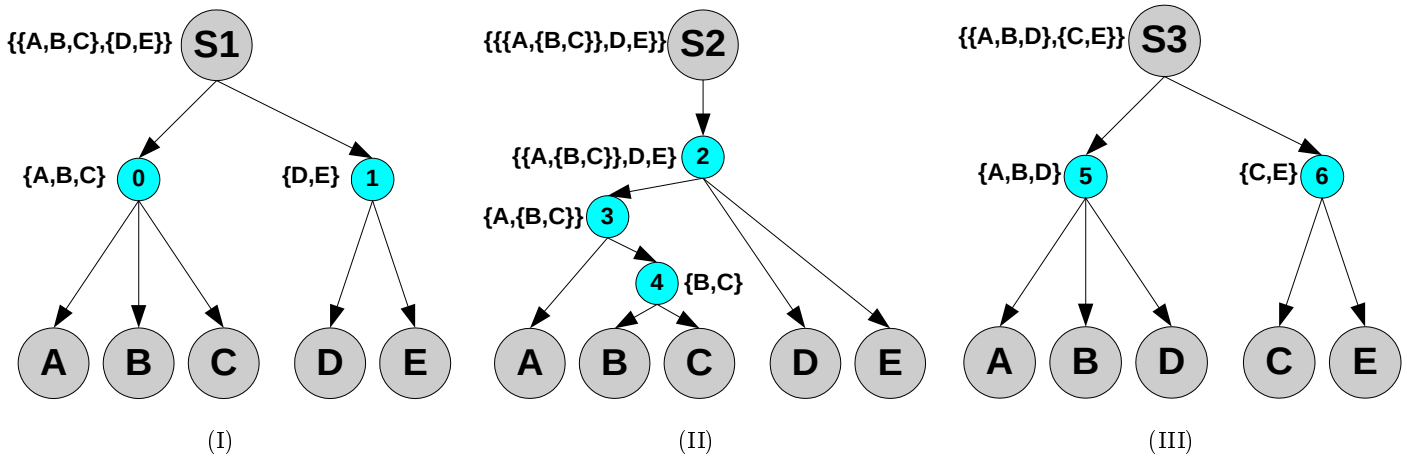


Figure 9: Illustration of the representation of trees internal nodes as *equivalence classes* of destinations with equal shared metric values from the server. Also, an example of the application of the *reduction* and *compression* constructions (II)(III) to extend the minimal tree in (I) and detect inconsistent trees (III).

Appendix: Proofs

Lemma 1 *The minimal tree T' spanning two mutually consistent topologies $\mathcal{T}_1$ and $\mathcal{T}_2$ from the same server to a set of m clients is unique and can be constructed in the worst case in $O(m^2.log(d))$ time where d is the maximum depth in any of the trees.*

Proof: We prove the lemma by providing a construction that uses $\mathcal{T}_1$ and $\mathcal{T}_2$ to generate the unique T', prove the correctness of the construction and verify that its running time is $O(m^2.log(d))$. To simplify our construction description, we start by describing the data structures that we require to represent the trees.[11]

One can think of each internal node in a tree $\mathcal{T}$ as an equivalence class of destinations (or destination classes) with equal shared metric values from the server. Figure 9 illustrates the equivalence classes concept. As an example, in figure 9(I) node 0 equivalence class {A,B,C} signifies that the shared metric value between any pair of destinations in the set {A,B,C} is the same. The reason is that the shared path between all pairs in the set is the same (path $(S-0)$). In figure 9(II) node 0 equivalence class {A,{B,C}} signifies that the shared metric value between destination A and either of the destinations B and C is the same but the sharing between destinations B and C is more observable because of the additional path they share (path $(0-1)$). This is symbolically equivalent to the equivalence class {A,1} where 1 is the identity of the internal node represented by the equivalence class {B,C}. Obviously, the equivalence class of the root node of a tree (node S) gives a full description of the tree.

The input to the minimal tree construction consists of two lists T_1 and T_2 of the equivalence classes corresponding to the internal nodes (including the root nodes) of each of the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ ordered in descending order of nodes depth. To avoid confusion we assign different identifiers to different internal nodes in different trees. The minimal tree construction makes use of two other lists L_1 and L_2. These lists contain the paths from the root node to each node in the trees indexed by nodes identifiers[12]. As an example if tree (I) in figure 9 is $\mathcal{T}_1$ and tree (II) is $\mathcal{T}_2$ then the input to the minimal tree construction would be the

[11] If trees are represented differently, a pass over the trees nodes to adapt the representation still takes $O(m)$.

[12] The indexing of the lists by nodes identifiers avoids searching the trees nodes for nodes equivalence classes.

T_1

0	**{A,B,C}**
1	**{D,E}**
S1	**{0,1}**

T_2

4	**{B,C}**
3	**{A,4}**
2	**{3,D,E}**
S2	**{2}**

L_1

A	**S1,0**
B	**S1,0**
C	**S1,0**
D	**S1,1**
E	**S1,1**
0	**S1**
1	**S1**
S1	**NULL**

L_2

A	**S2,2,3**
B	**S2,2,3,4**
C	**S2,2,3,4**
D	**S2,2**
E	**S2,2**
2	**S2**
3	**S2,2**
4	**S2,2,3**
S2	**NULL**

Figure 10: Data Structures Used in the Minimal Tree Construction for Trees (I) and (II) of Figure 9.

lists T_1, T_2, L_1 and L_2 shown in figure 10. To complete the notation we use in the construction, let n_{C_N} be the number of elements of the equivalence class of node N and let n_{D_N} be the number of destinations that are descendents of node N. for example, by referring to figure 9(II) $n_{C_2} = 3$ and $n_{D_2} = 5$. Note that both n_{C_N} and n_{D_N} could be computed recursively in $O(m)$ time.

Figure 11 gives the details of the proposed minimal topology construction technique. It should be a simple exercise to verify that feeding tree (I) and tree (II) of figure 9 to the `MinimalTopologyConstruction` function leads to the tree (II) after inserting a new edge between node 2 and the destinations D and E, while feeding tree (I) and tree (III) of figure 9 to the same function leads to an inconsistency alarm.

We next verify that the *running time* bound for the construction is $O(m^2.log(d))$ where m is the number of destinations (clients) and d is the maximum depth in any of the two trees. Constructing L_1, L_2, n_{C_N} and n_{D_N} for all nodes is basically a path over all trees nodes which could be done recursively in $O(m)$ since the number of internal nodes of a tree is $O(m)$. As shown in figure 11, the construction is a loop over all nodes in T_2 ($O(m)$ of such nodes), and for each of these nodes there can be, in addition to a constant number of comparisons, a comparison between x elements of the L_1 list where x is the degree of a tree node , and each element to be compared is of length y where y is the depth of the childs of this node. Such comparison take $x.log(y)$ time because each of the x elements is sorted. x is at most m and y is at most d and so the comparison is $O(m.log(d))$ time in the worst case. The overall running time of the construction in the worst case is thus $O(m^2.log(d))$.

Now, we will prove the correctness of the proposed minimal topology construction technique. The construction uses T_1 as a potential minimal tree and checks the equivalence class of every node in T_2 for consistency with T_1 nodes equivalence classes and identifies whether it adds links to the potential minimal tree T_1 or not. In checking a T_2 node equivalence class against T_1, the construction relies on the fact if an element of an equivalence experienced a certain metric value then, by definition of an equivalence class, all the elements in the same equivalence class should always experience the same metric value or else we have an inconsistency. Also, checking T_2 internal nodes in descending order of their depth the construction ensures that every internal node to be checked has its descendents already checked. If N is the T_2 internal node to be checked then Without loss of generality, there are four possible scenarios:

1. **Exact Match:** In this case N has a twin node in T_1 and thus no new nodes need to be inserted. The construction identifies this case by checking that node N childs in T_1 are all the only childs of the same parent node.

2. **Reduction:** In this case node N identifies an extra level of sharing that does not appear in T_1. As an example node 4 in figure 9 adds another level of sharing to T_1 node 0. That is a link should be

```
Function MinimalTopologyConstruction(T1,T2) {
   Construct L1, L2, nC and nD from T1 and T2;
   For (all nodes N in T2 in decreasing order of their depth) do {
      if ((nD_N == 1)AND(nD_S1 > 1)) then
         Add a link in T1 between S1 and its childs;
      else
         if (all L1[i] ∀i ∈ CN are equal) then
            Let w be the last element in L1[i] for any one i ∈ CN;
            if (nC_N < nC_w) then                          /* Reduction */
               Add node N and link between node w and node N in T1;
               Insert an entry L1[N] = L1[i] for any one i ∈ CN;
               Let nC_w = nC_w − nC_N + 1;
            else                                           /* Exact Match */
               Insert an entry L1[N] = L1[i] for any i ∈ CN after removing w;
            fi
         else
            Let w be the last matching element in all L1[i] ∀i ∈ CN;
            if (nD_N == nD_w) then                         /* Compression */
               Insert an entry L1[N] = prefix of L1[i] upto before w for any i ∈ CN;
            else                                           /* Inconsistency */
               Signal the presence of an inconsistent topology;
            fi
         fi
      fi
   }
   return ( T1 );              /* T1 now is the minimal tree of old T1 and T2 */
}
```

Figure 11: Minimal Topology Construction Technique.

inserted between node 0 and its childs B and C. The construction identifies this case by checking that node N childs in T_1 are childs of the same parent but not the only childs of this parent.

3. **Compression:** In this case node N does *not* identify an extra level of sharing that T_1 identifies. As an example node 2 in figure 9 does not catch the level of sharing that is between nodes D and E in T_1. This case though consistent does not enhance the potential minimal tree T_1. The construction identifies this case by checking that not all node N childs are children of the same parent in T_1 and that the deepest internal node in T_1 that has node N childs as its descendents (node w) is connected to the same number of clients as the number of node N connected clients. The intuition behind checking the number of clients connected to node w in T_1 and the number of clients connected to node N in T_2 is that if the trees are consistent and this is indeed a compression then node w clients should all match node N clients. By construction, all the clients connected to node N in T_2 are also connected to node w in T_1. So if the number of clients connected to node N and the number of clients connected to node w match then the clients must also match.

4. **Inconsistency:** In this case node N childs are in different equivalence classes in T_1 and are not the only childs of their deepest common parent in T_1. As an example nodes 6 and 5 in tree (III) of figure 9 are inconsistent with trees (I) and (II) of the same figure. The construction identifies this case by checking that not all node N childs in T_1 are children of the same parent and that the deepest internal node in T_1 that has node N childs as its descendents (node w) is connected to different number of clients than the number of node N connected clients.

■

Theorem 2 *The minimal tree T' spanning a set of mutually consistent topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ connecting a server to a set of m clients is unique and can be constructed in $O(n.m^2.log(d))$ time.*

Proof: The proof is by induction over the set of mutually consistent topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$

Induction Base: ($n = 1$) Given only one topology $\mathcal{T}_1$, the minimal tree spanning $\mathcal{T}_1$ ($T'_{n=1}$) is $\mathcal{T}_1$ itself. Obviously, $T'_{n=1}$ is unique and obatining $T'_{n=1}$ given $\mathcal{T}_1$ is $O(m^2.log(d))$ time.

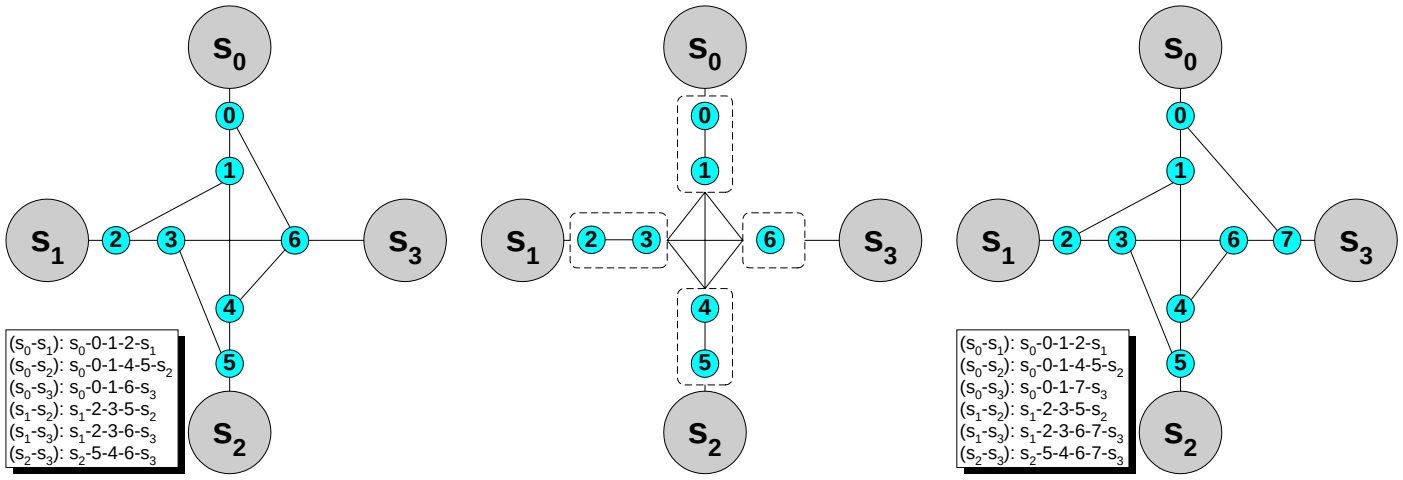
Induction Hypothesis: ($n = k - 1$) Assume that the minimal tree $T'_{n=k-1}$ spanning a set of $k-1$ mutually consistent topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{k-1}$ is unique and can be constructed in $O(m^2.log(d))$ time.

Induction Step: ($n = k$) Given a set of k mutually consistent topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, we will show that the minimal tree $T'_{n=k}$ spanning the trees in this set is unique and can be constructed in $O(m^2.log(d))$ time. From the induction hypothesis, we assume we can get the unique $T'_{n=k-1}$ spanning the first $k-1$ trees in $O(m^2.log(d))$ time. Feeding $T'_{n=k-1}$ and $\mathcal{T}_k$ as input to the `MinimalTopologyConstruction` function we get $T'_{n=k}$ in $O(m^2.log(d))$ time (lemma 1). The total running time is thus in the worst case $O(n.m^2.log(d))$.

■

Theorem 3 *Given a set of n sources s_0, s_1, ..., s_{n-1} and a procedure that enables the evaluation of $f(p_{s_i})$, $f(p_{s_j})$, and $f(p_{s_i.s_j})$ for some monotone, separable and symmetric metric f between any source s_k and any two other sources s_i and s_j, one can efficiently* infer *and* label *the base topology $\mathcal{G}$ induced by f over the physical topology induced through IP routing and connecting the n sources for any sensitivity parameter $c > 0$.*

Proof: We next provide a proof by construction of the theorem together with the reasoning behind the construction. Our inference of $\mathcal{G}$ is based on the efficient inference of $\mathcal{T}_i$ $\forall i = 0, 1, \ldots, n-1$ the base topology induced by f over the physical topology connecting source s_i to the other $n-1$ sources (theorem 1). Based on the proof of theorem 1 the structure of $\mathcal{T}_i$ is a tree rooted at s_i, a path (s_i, r_i) connects s_i to an internal node r_i from which the paths to the sources with the highest shared metric value diverges and paths to the other sources diverge from different points along the path (s_i, r_i) ordered in increasing order of shared metric values. Since the physical topology connecting the sources is induced by IP routing protocol, the path connecting a source s_i to another s_j is the same path connecting s_j to s_i (in the reverse direction). This means that for any pair of sources s_i and s_j, the branching point $b_{i,j}$ from (s_i, r_i) going to s_j should be physically connected to the branching point $b_{j,i}$ from (s_j, r_j) to s_i. The path $(b_{i,j}, b_{j,i})$ should appear in $\mathcal{G}$ iff



(A) Sample Graph (B) Virtual Clique (C) Sample Minimal General Graph

Figure 12: A sample graph connecting four servers s_0, s_1, s_2 and s_3 (I), with an illustration of the *virtual clique* connecting the servers (II) and a sample minimal general graph (III).

$f((b_{i,j}, b_{j,i})) > c$. The question becomes: Can we infer $f((b_{i,j}, b_{j,i}))$ from end-to-end measurements ? The assumption is that there is a procedure that enables the evaluation of $f((s_i, b_{i,j}))$, $f((s_i, s_j))$, $f((s_j, b_{j,i}))$ and $f(s_j, s_i)$. From the proof of theorem 1, monotonicity and separability of the metric f ensures that we can also estimate $g(f(s_i, s_j), f((s_i, b_{i,j}))) = f((b_{i,j}, s_j))$ and $g(f(s_j, s_i), f((s_j, b_{j,i}))) = f((b_{j,i}, s_i))$. Note that $f((b_{i,j}, b_{j,i})) = g(f((b_{i,j}, s_j)), f((b_{j,i}), s_j)))$ and $f((b_{j,i}, b_{i,j})) = g(f((b_{j,i}, s_i)), f((b_{i,j}), s_i)))$. If f is also symmetric we could rewrite the above equations as $f((b_{i,j}, b_{j,i})) = f((b_{j,i}, b_{i,j})) = g(f((b_{i,j}, s_j)), f((s_j, b_{j,i})))$ and thus we can get an estimate for $f((b_{i,j}, b_{j,i}))$ (or equivalently for $f((b_{j,i}, b_{i,j}))$). If $f((b_{i,j}, b_{j,i})) > c$ then we include an $(b_{i,j}, b_{j,i})$ edge in $\mathcal{G}$. The labeling of all $\mathcal{G}$ edges is straightforward given $f((s_i, b_{i,j}))$, $f((b_{i,j}, b_{j,i}))$ and $f((b_{j,i}, s_j))$ between every pair of sources s_i and s_j. ■

Theorem 4 *The minimal general graph $\mathcal{G}$ connecting a set of n sources s_0, s_1, ..., s_{n-1} through IP routing has $\frac{3.n.(n-1)}{2} - n$ segments.*

Proof: By "minimal general" graph we mean that removing any edge from $\mathcal{G}$ will make the resulting graph unable to represent some of the physical grpahs connecting n sources through IP routing. Note that the statement of this theorem does not mean that any graph with $\frac{3.n.(n-1)}{2}$ segments is the minimal general graph. Let $d_{i,j}$ be the internal node along the path (s_i, r_i), that has a branching path to source s_j $i \neq j$ AND $(d_{i,j}, s_j)$ does not intersect (s_i, r_i) except at node $d_{i,j}$. The proof of theorem 3 suggests that the graph between n sources is a *virtual* clique of degree n connecting every internal node $d_{i,j}$ to node $d_{j,i}$.

Figure 12 illustrates the virtual clique graph connecting four servers s_0, s_1, s_2 and s_3. Note that the graph in figure 12 (I) is not the minimal between any set of four sources. The reason being that the tree connecting source s_3 to the other sources is not a binary tree. Binary trees have the most internal nodes and thus most segments. Figure 12 (III) shows the corresponding minimal graph for this set of four servers and since it consists of binary trees its has the same number of segments as the minimal graph between any set of four servers. The number of segments in the minimal graph between any set of four servers is thus $4.(4-2) + 4.(4-1)/2$ and in general the number of segments in a minimal graph connecting any set of n servers is $n.(n-2) + \frac{n.(n-1)}{2} = \frac{3.n.(n-1)}{2} - n$. ■

References

[1] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *Proceedings of ACM SIGCOMM '99*, 1999.

[2] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture For Internet Hosts. In *SIGCOMM '99*, Cambridge, MA, September 1999.

[3] J. C. Bolot. End-to-end Packet Delay and Loss Behavior in the Internet. In *SIGCOMM '93*, pages 289–298, September 1993.

[4] J. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proceedings of IEEE INFOCOM '99*, pages 275–83, March 1999.

[5] R. Cáceres, N. G. Duffield, J. Horowitz, D. Towsley, and T. Bu. Multicast Based Inference of Network-Internal Characteristics: Accuracy of Packet-Loss Estimation. In *INFOCOM '99*, March 1999.

[6] R. Cáceres, N. G. Duffield, S. B. Moon, and D. Towsley. Inference of Internal Loss Rates in the MBone. In *IEEE Global Internet (Globecom)*, Rio de Janeiro, Brazil, 1999.

[7] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of 1997 USENIX Symposium on Internet Technology and Systems*, Dec 1997.

[8] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide area networks. In *Proceedings of IEEE INFOCOM '97*, April 1997.

[9] R. Carter and M. E. Crovella. Measuring bottleneck link speed in packet switched networks. In *PERFORMANCE '96, the International Conference on Performance Theory, Measurement and Evaluation of Computer and Communication Systems*, October 1996.

[10] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *In the Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[11] M. Coates and R. Nowak. Network Loss Inference Using Unicast End-to-End Measurement. In *Proceedings of ITC Conference on IP Traffic, Modelling and Management*, Monterey, CA, September 2000.

[12] N. Duffield, V. Paxson, and D. Towsley. MINC: Multicast-based Inference of Network-Internal Characteristics. http://www-net.cs.umass.edu/minc/.

[13] N. Duffield, F. Lo Presti, V. Paxson, and D. Towsley. Inferring Link Loss Using Striped Unicast Probes. In *IEEE INFOCOM 2001*, April 2001.

[14] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of ACM SIGCOMM '99*, September 1999.

[15] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of IEEE INFOCOM '98*, April 1998.

[16] Cooperative Association for Internet Data Analysis (CAIDA). The Skitter project. http://www.caida.org/Tools/Skitter.

[17] R. Govindan and A. Reddy. An analysis of internet inter-domain routing and route stability. In *Proceedings of IEEE INFOCOM '97*, April 1997.

[18] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *Proceedings of IEEE INFOCOM '00*, April 2000.

[19] T. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. In *ACM SIGCOMM*, pages 277–88, Cambridge, MA, September 1999.

[20] James Gwertzman and Margo Seltzer. The case for geographical push caching. In *Proceedings of HotOS'95: The Fifth IEEE Workshop on Hot Topics in Operating Systems*, Washington, May 1995.

[21] K. Harfoush, A. Bestavros, and J. Byers. Robust Identification of Shared Losses Using End-to-End Unicast Probes. In *Proceedings of the 8th Annual International Conference on Network Protocols (ICNP)*, Osaka, Japan, November 2000.

[22] IPMA : Internet Performance Measurement and Analysis. http://www.merit.edu/ipma.

[23] V. Jacobson. `Pathchar`: A Tool to Infer Characteristics of Internet Paths. ftp://ftp.ee.lbl.gov/pathchar.

[24] S. Keshav. *Congestion Control in Computer Networks.* PhD thesis, University of California at Berkeley, September 1991.

[25] Kevin Lai and Mary Baker. Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. In *SIGCOMM' 00*, August 2000.

[26] Kevin Lai and Mary Baker. Nettimer: A tool for Measuring Bottleneck Link Bandwidth. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, March 2001.

[27] Mtrace: Tracing multicast path between a source and a receiver. ftp://ftp.parc.xerox.com/pub/netsearch/ipmulti.

[28] Venkata N. Padmanabhan. *Addressing the challenges of Web Data Transport.* PhD thesis, University of California at Berkeley, September 1998.

[29] J.-J. Pansiot and D. Grad. On Routes and Multicast Trees in the Internet. *Computer Communication Review*, 28(1):41–50, January 1998.

[30] V. Paxson. End-to-end Routing Behavior in the Internet. In *SIGCOMM '96*, Stanford, California, August 1996.

[31] V. Paxson. End-to-End Internet Packet Dynamics. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[32] V. Paxson. *Measurements and Analysis of End-to-end Internet Dynamics.* PhD thesis, U.C. Berkeley and Lawrence Berkeley Laboratory, 1997.

[33] Internet Mapping Project. http://www.cs.belllabs.com/who/ches/map/, 1999.

[34] The SCAN Project. http://www.isi.edu/scan/, 1999.

[35] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. Technical Report IETF RFC 2481, January 1999. Available at `ftp://ftp.isi.edu/in-notes/rfc2481.txt`.

[36] S. Ratnasamy and S. McCanne. Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements. In *Proceedings of IEEE INFOCOM '99*, pages 353–60, March 1999.

[37] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows Via End-to-end Measurement. In *ACM SIGMETRICS '00*, Santa Clara, Ca, June 2000.

[38] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of web documents. In *Proceedings of the 6th International WWW Conference*, 1997.

[39] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc. of Usenix Symposium on Internet Technologies and Systems (USITS) '97*, Monterey, CA, December 1997.

[40] Akamai Technologies. Freeflow content delivery system. http://www.akamai.com.

[41] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *Proceedings of IEEE INFOCOM '99*, pages 345–52, March 1999.