Inference and Labeling of Metric-Induced Network Topologies

Azer Bestavros, *Member*, *IEEE*, John W. Byers, and Khaled A. Harfoush, *Member*, *IEEE*

Abstract—The development and deployment of distributed network-aware applications and services 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 to be general-purpose, should enable a representation of properties along temporal, spatial, and measurement resolution dimensions. In this paper, we propose MINT—a general framework for the construction of such metric-induced models using end-to-end measurements. We present the basic theoretical underpinnings of MINT for a broad class of performance metrics, and describe PERISCOPE, a Linux embodiment of MINT constructions. We instantiate MINT and PERISCOPE for a specific metric of interest—namely, packet loss rates—and present results of simulations and Internet measurements that confirm the effectiveness and robustness of our constructions over a wide range of network conditions.

Index Terms—End-to-end measurement, packet-pair probing, Bayesian probing, Internet tomography, performance evaluation.

1 INTRODUCTION

MANY of today's Internet services are administered through colocation facilities that are distributed over the wide area. Examples include Content Distribution Networks (CDNs) and Application Service Providers (ASPs). Internet servers at such facilities typically command a large number of concurrent unicast connections. We use the term *Mass Servers* to refer to such *Mass*ively Accessed *Servers*. A collection of Mass servers distributed over a wide-area network 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 the context of a CDN, such a "map" of network conditions could be used for server selection or to inform replica placement decisions. In the context of grid computing, such information could be used to inform code versus data movement, for example. In order for such *network-aware strategies* to be practical, an endpoint—whether a CDN caching proxy or a grid host—must be able to accurately infer the conditions of network resources connecting it to other endpoints. Typically, these conditions are measured in terms of one or more performance metrics of interest, such as available bandwidth, hop-count, loss rate, delay, and jitter.

Previous studies aimed at the identification of network-internal conditions have focused on the characterization of physical network entities (e.g., routers, links, AS boundaries). For the resource management problems that face Mass servers, an accurate characterization of *all* physical resources between the server and its clients is neither necessary nor feasible. Instead, an abstraction that captures the conditions of only those shared resources that need to be accounted for and judiciously managed is preferable. For a streaming media delivery application, a model of the network that captures path *bottleneck bandwidth* and *jitter* might be sufficient for server selection purposes. Alternatively, for a distributed caching or content distribution application, a model of the network that captures *loss rates* on congested paths may be used for optimizing replica placement.

End-to-end measurements made in the course of normal operations by a Mass server provide a wealth of information about the end-to-end characteristics of a particular 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 [2]. In addition to these connection-specific statistics, end-to-end measurements correlated across different connections (at one or more Mass servers) can be used to infer conditions at a finer granularity than encompassing an entire path, and yet coarser than that at a specific network resource (e.g., for a portion of a path as opposed to for every hop along the path). This paper proposes a framework for efficient inference of network conditions for a wide range of metrics. In particular, it proposes a framework that enables a parametrized abstraction that summarizes the topology connecting a set of endpoints with respect to a given metric of interest.

1.1 Metric-Induced Network Topologies

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

- A. Bestavros and J.W. Byers are with the Department of Computer Science, Boston University, 111 Cummington Street, MCS-140E, Boston, MA 02215. E-mail: best@bu.edu, byers@cs.bu.edu.
- K.A. Harfoush is with the Department of Computer Science, North Carolina State University, 1010 Main Campus Dr., Box 7534, EGRC building, Room 457, Raleigh, NC 27695-7534. E-mail: harfoush@cs.ncsu.edu.

Manuscript received 5 Dec. 2003; revised 27 Nov. 2004; accepted 6 Dec. 2004; published online 21 Sept. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0227-1203.

by edges in this topology. The threshold c acts as a "compression parameter"—the larger the value of c, the coarser (i.e., less detailed) the resulting MINT graph will be. By hiding uninteresting details of a physical topology (e.g., sequences of links with abundant bandwidth, negligible jitter, or insignificant losses), MINT graphs enable a compact representation of those resources that need to be accounted for by network-aware applications at Mass servers. We refer to this compact representation as a *caricature* of the underlying network.

1.2 Paper Contributions and Overview

Existing techniques for making inferences from end-to-end observations are special-purpose, each targeting specific performance metrics of interest, such as estimation of bottleneck bandwidth, packet loss rate, propagation delay, or jitter. In this paper, we study the extent to which such end-to-end methods can be effective, not by considering individual specialized techniques, but by reasoning about properties of the performance metrics themselves. Our work introduces the concept of Metric-Induced Network Topologies (MINT) and proposes general procedures for the inference and labeling of MINT topologies, as well as the integration of MINT topologies obtained at different points in time and from different network vantage points. We present an embodiment of MINT procedures in Linux, in the form of a Linux API called PERISCOPE (a Probing Engine for the Recovery of Internet Subgraphs). We specify a broad class of performance metrics for which our proposed techniques are applicable, then instantiate MINT and PERISCOPE for one such metric, packet loss rates. In this case study, we demonstrate that recently proposed end-to-end techniques for the estimation of shared losses can be leveraged to enable the characterization of *loss topologies*. To do so, we present results both of extensive simulations and of Internet deployment results that demonstrate the accuracy and convergence of our loss topology inference and labeling techniques.

2 Related Work

2.1 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], [23] or to perform Internet topology characterization [16], [17], [31]. 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, applicable at shorter time scales, is statistical inference of network internal characteristics based on end-to-end measurements of point-to-point traffic. Although this strategy has been employed in networking contexts for at least a decade, starting with the use of packet-pair probes [24], [4], recent efforts have greatly expanded the set of available techniques [1], [5], [8], [9], [14], [21], [27], [33], [35], [36]. The diversity of inference techniques cut across approaches which are sender-driven, receiver-driven, those which employ multicast and those which employ unicast probes, and span performance metrics ranging from static metrics (e.g., bottleneck bandwidth) to dynamic metrics (e.g., packet loss rates). However, none of these studies attempt to provide a *general* framework to describe when inferences may be drawn. Our approach does so by focusing not on the specific probing technique, but by concentrating on the metrics of interest, and by developing an understanding of which metrics are amenable to inference techniques.

2.2 Estimation of Shared Loss Using End-to-End Measurements

The specific problem of inferring and labeling loss topologies is motivated in part by recent work on topological inference over *multicast* sessions [6], [33], [11], [12]. By making purely end-to-end observations of packet loss at endpoints of multicast sessions, Ratnasamy and McCanne [33] and Cáceres et al. [6] have demonstrated how to make unbiased, maximum likelihood estimation inferences of 1) the multicast tree topology and 2) the packet loss rates on the edges of the tree, respectively.

At the core of our methodology for constructing loss topologies is the need for a procedure for the measurement of shared losses between one server and multiple destinations. A number of recent efforts have addressed this problem; all would be suitable as underlying procedures in our framework. In [34], Rubenstein et al. 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 [19], we presented an alternative technique for the identification of shared losses using a Bayesian Probing (BP) approach. Using BP, 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. Other similar techniques have been recently proposed [13], [10], [25], [26]. For example, the striped unicast probing of Duffield et al. 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 [13].

3 Metric-Induced Network Topologies

3.1 Basic Definitions

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. Thus, effective methods for producing a metric-induced topology must be *sensitive* to different possible thresholds for an observable value of the metric. The framework we present in this section and the PERISCOPE tool we present in Section 4 therefore provide a multiscale representation of a metric-induced topology.

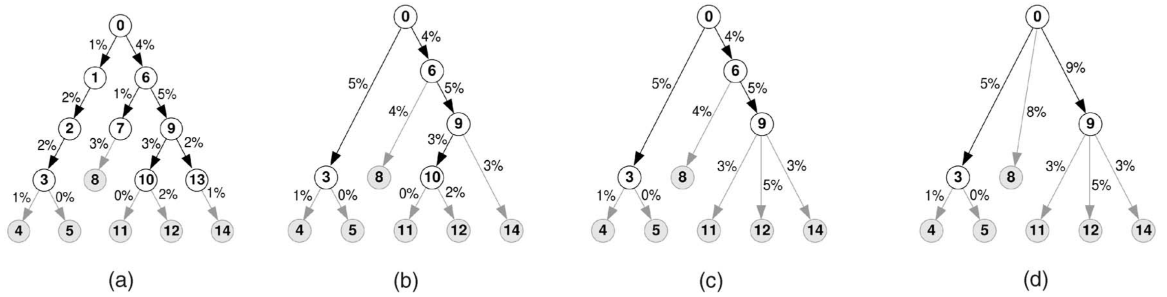

Fig. 1. Relation between physical and loss topologies under various sensitivity parameter c. Gray links are edges to leaf nodes. (a) Physical tree, (b) loss tree ($c = 0$), (c) loss tree ($0.03 < c \leq 0.04$), and (d) loss tree ($0.04 < c \leq 0.05$).

Beginning with basic definitions, we formalize these notions below:

Consider the set of links used to route unicast traffic between a server and a set of clients. In an idealized routing environment, 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. Across this idealized tree, 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. In practice, realities of routing do not guarantee that a set of routes from a server form a tree since routes to two remote hosts can separate and rejoin before diverging again, load balancers can employ multipath routing, and faulty behavior such as route flapping can cause packets to a single host to traverse different paths. Our methods are not immediately applicable to paths using load balancers or where route flapping is occurring, but our methods do apply to paths which diverge and rejoin. However, for such paths, we will treat them as paths which diverge and do not rejoin. This enables us to model such paths in an idealized tree framework, but implies that we will not be able to accurately capture the incidence of sharing beyond the first point of divergence in the paths. Addressing this question appears difficult, and is considered in our future work. For the remainder of the paper, we assume that a tree model of the network to be characterized is appropriate.

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

We say that we *collapse* a node i of a tree into its parent j if we delete the edge (i, j) from the tree and attach all children of i to j, by replacing all edges (i, k) with edges (j, k). Note that when this operation is applied on 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.*

We now extend our definitions to apply to topologies with edges labeled according to a metric, noting that, in general, we do not assume knowledge of the underlying physical or logical topology. 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 multihop physical paths as well as physical links. For some metrics, end-to-end observations will have difficulty distinguishing a small nonzero metrical value from a zero value. As a result, methods may then misclassify incidence of sharing. For this reason, we parameterize 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, $\mathcal{T}_c(f)$, is the labeled 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.*[1]

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 $\mathcal{T}_c$. We refer to the tree $\mathcal{T}_0$ as the *base topology*, as this is the condensed representation that reflects only the edges of the logical topology that have a nonzero metrical value. Increasing the value of the sensitivity parameter c to create trees $\mathcal{T}_c$ generate 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 Fig. 1a. 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 Fig. 1a are labeled with the actual loss rates on these links. Fig. 1b 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. Figs. 1c and 1d show the loss

1. 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.

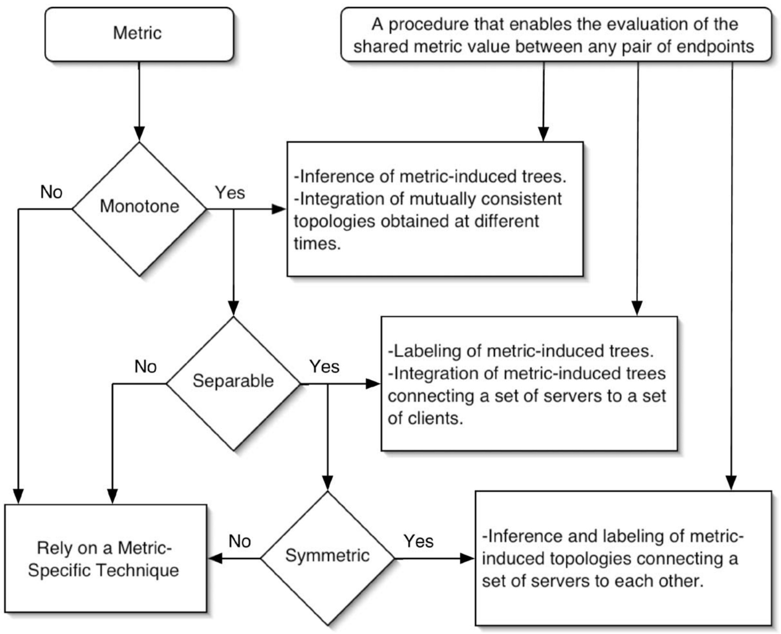
Metric
A procedure that enables the evaluation of the shared metric value between any pair of endpoints
Monotone
No
Yes
-Inference of metric-induced trees.
-Integration of mutually consistent topologies obtained at different times.
Separable
No
Yes
-Labeling of metric-induced trees.
-Integration of metric-induced trees connecting a set of servers to a set of clients.
Rely on a Metric-Specific Technique
No
Symmetric
Yes
-Inference and labeling of metric-induced topologies connecting a set of servers to each other.

Fig. 2. Summary of MINT framework functionality.

topologies for this physical tree when $0.03 < c \leq 0.04$ and when $0.04 < c \leq 0.05$, respectively. 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 (at the cost of a small degree of inaccuracy).

3.2 MINT Inference and Labeling

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

To define these conditions, we need a bit more terminology. We denote a path between two hosts A and B by $p_{A \to B}$ and denote a general path between any two end hosts as p_i, where the subscript represents a path identifier. 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. We classify metrics based on three different properties:

- **Monotonicity**: A metric f is *monotonically increasing* if for any pair of paths p_i, p_j for which p_i is a subpath of p_j, $f(p_i) < f(p_j)$. Similar definitions hold for metrics which are monotonically nondecreasing, monotonically decreasing, and monotonically nonincreasing, and we refer to any such metric as *monotone*.
- **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 the value of one subpath is determined by the values on the full path and the value on the peer subpath.
- **Symmetry**: A metric f is *symmetric* if for any path p_i the value $f(p_i)$ is equal as observed from either end of p_i.

Metrics of interest are diverse with respect to these three properties. For example, loss rate is monotone, separable but not symmetric; propagation delay is monotone, separable and (sometimes) symmetric; round-trip time is monotone, separable, and symmetric; available bandwidth is monotone, but not necessarily symmetric and definitely not separable; and jitter is neither monotone, nor separable, nor symmetric. Fig. 2 summarizes the MINT framework capabilities in the presence (or absence) of the monotonicity, separability, and symmetry properties. We will now show how the presence (or absence) of each these properties in a metric f affect our ability to infer, label, and aggregate topologies induced by f using end-to-end measurements. We do so through a sequence of theorems.

Theorem 1. *Given a procedure that evaluates $f(p_{s_0 \to a})$, $f(p_{s_0 \to b})$, and $f(p_{s_0 \to a . s_0 \to b})$ for some monotone metric f between a server s_0 and any two nodes a and b, one can* infer *the base topology induced by f over an arbitrary physical topology $\mathcal{T}$ in polynomial time. In the event that f is separable, one can also* label *the topology induced by f for any sensitivity parameter $c > 0$ in polynomial time.*

Proof (Sketch). We first demonstrate how to recursively infer the base topology $\mathcal{T}$, based on end-to-end evaluations of a monotonically increasing function f. Consider a set of n endpoints $s_1, s_2, \ldots s_n$. Sort all pairs of endpoints (s_u, s_v) on the value of $f(p_{s_0 \to s_u . s_0 \to s_v})$, and let

s_{i1} and s_{i2} be one such pair of endpoints which obtains the maximal value. Of the endpoints in the list, s_{i1} and s_{i2} have a maximal degree of sharing and, thus, there exists an internal node r in $\mathcal{T}_{\prime}$ at which the paths from s_{i1} and s_{i2} diverge, but between r and s_{i1} and between r and s_{i2}, there are no additional branches in $\mathcal{T}_{\prime}$. We prove this latter claim by contradiction. Suppose there were a branch at node q between r and s_{i1}, branching between s_{i1} and a third endpoint s_{i3}. Then, by monotonicity, $f(p_{s_0 \rightarrow s_{i1}.s_0 \rightarrow s_{i3}}) > (p_{s_0 \rightarrow s_{i1}.s_0 \rightarrow s_{i2}})$, which in turn violates maximality and leads to a contradiction. At this point, we have identified an internal node r and two of its children, but we have not identified all of its children. To do so, note that for each of r's children s_c, we must have that $f(p_{s_0 \rightarrow s_{i1}.s_0 \rightarrow s_c}) = (p_{s_0 \rightarrow s_{i1}.s_0 \rightarrow s_{i2}})$. So, to locate all of r's children, we simply scan the set of endpoint pairs whose shared path value was maximum, and include all those meeting the criterion above. Note that we may bypass those node pairs (s_u, s_v) whose value also happens to be maximal, but who are unrelated to r's children.

The above construction identifies an internal node r in the logical topology, and gives a method for computing $f(p_{s_0 \rightarrow r}) = f(p_{s_0 \rightarrow s_i.s_0 \rightarrow s_j})$, and identifying all of r's direct descendants in the base topology. Therefore, we can remove s_i, s_j and all other children of r from consideration and replace them with the single node r. This reduces 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 repeatedly, eventually yields 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 $g(f(p_{s_0 \rightarrow r_i}), f(p_{s_0 \rightarrow r_j})) < c$. (Note that the path to the child is the full path, and the path to the parent is the subpath.) 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, the updated value is $g(f(p_{s_0 \rightarrow r_i}), f(p_{s_0 \rightarrow r_j}))$. The result is a labeled topology as depicted in Fig. 1. □

An immediate corollary to the theorem above is that inference and labeling can be conducted between a set of servers and a single endpoint provided the metric is monotone and separable.

3.3 Integrating MINT Snapshots

For nonstationary metrics such as loss rates and available bandwidth, the labels 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 both of these reasons, 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 an inferred (but unlabelled) topology which includes the *union* of all internal edges observed in the snapshots. A dynamic picture of the MINT snapshots over time may be useful in providing the temporal evolution of the bottlenecks. It is natural to consider whether one can efficiently produce such an integrated topology from a set of mutually consistent topological snapshots. The following theorem shows that this can be achieved.

Definition 4. *Consider a set of unlabeled metric-induced tree topologies $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ rooted at a node x and spanning destination sets $E_1, E_2, \ldots E_k$, respectively. These trees are* mutually consistent *if there exists a tree $\mathcal{T}'$ spanning all destinations in $\bigcup_i E_i$ from which for any i, $\mathcal{T}_i$ can be generated from $\mathcal{T}'$ by repeated collapse operations.*

Theorem 2. *The minimal tree $\mathcal{T}'$ 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 $O(nm^2 \log(m))$ time.*

Proof. In order to prove the theorem, we first demonstrate that the minimal tree spanning any *two* mutually consistent topologies is unique and can be constructed in O($m^2 \log m$) time. The construction which we present for merging two trees is associative with respect to preserving minimality and uniqueness, thus it can be applied $n - 1$ times in succession: first, to merge $\mathcal{T}_1$ and $\mathcal{T}_2$, then between the resulting unique minimal tree and $\mathcal{T}_3$, etc. The overall running time is then O($nm^2 \log(m)$).□

We now prove the lemma that two mutually consistent topologies can be merged efficiently.

Lemma 1. *The minimal tree $\mathcal{T}'$ spanning two mutually consistent topologies $\mathcal{T}_1$ and $\mathcal{T}_2$ from a single server to a set of m clients is unique and can be constructed in $O(m^2 \log(m))$ time.*

Proof. We first prove uniqueness by demonstrating necessary conditions for nodes to be present in $\mathcal{T}'$. First, all leaf nodes (the m clients) and the root must necessarily be present. Furthermore, an internal node corresponding to each branch point in either $\mathcal{T}_1$ or $\mathcal{T}_2$ must be present in $\mathcal{T}'$, since collapse operations never remove branch points. In particular, let $leaves_{\mathcal{T}}(x)$ denote the set of leaves in the subtree rooted at node x in $\mathcal{T}$. Then, let $\mathcal{L}(\mathcal{T}', x) = leaves_{\mathcal{T}_1}(x) \bigcup leaves_{\mathcal{T}_2}(x)$, and $\mathcal{L}(\mathcal{T}') = \bigcup_{x \in \mathcal{T}_1 or x \in \mathcal{T}_2} \mathcal{L}(\mathcal{T}', x)$, e.g., the set of all distinct subsets of clients in a rooted subtree of $\mathcal{T}_1$ or $\mathcal{T}_2$. In the final tree $\mathcal{T}'$ we will have a node for each element of $\mathcal{L}(\mathcal{T}')$, noting that such a node may correspond to branch points in *both* $\mathcal{T}_1$ and $\mathcal{T}_2$. This set of nodes is minimal, and can be formed into a tree since the two topologies are assumed to be mutually consistent. Moreover, this tree is unique since node x is an ancestor of node y if and only if $leaves_{\mathcal{T}}(y) \subseteq leaves_{\mathcal{T}}(x)$, thus the node placement in the tree $\mathcal{T}$ is uniquely determined.

We now give a deterministic bottom-up construction to build this tree in O($m^2 \log(m)$) time. We start with some additional terminology. Let $parent_{\mathcal{T}}(x)$ be the parent of node x in tree $\mathcal{T}$, let $children_{\mathcal{T}}(x)$ be the set of nodes that are direct descendants of node x, and let $leaves_{\mathcal{T}}(x)$ again be

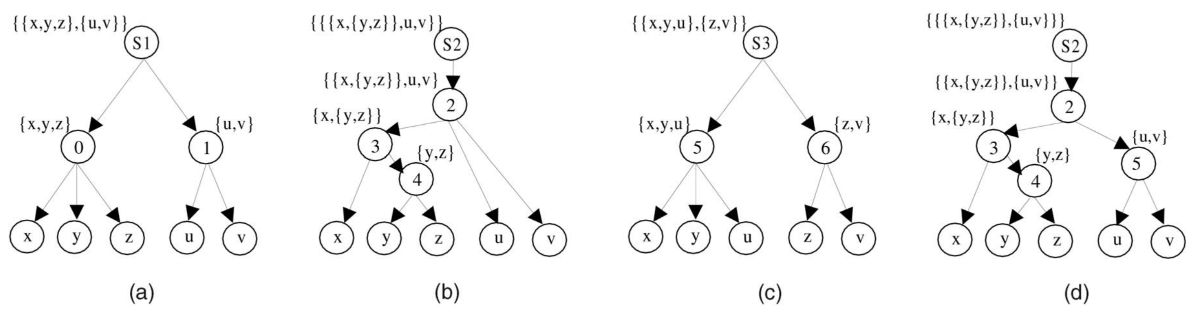

Fig. 3. Sample trees used to illustrate the minimal tree construction technique.

the set of leaves that are in the subtree rooted at node x. Finally, let $ancestor_{\mathcal{T}}(\mathcal{S})$ be the least common ancestor of all the nodes in the set $\mathcal{S}$. For example, by referring to Fig. 3b, $parent_{\mathcal{T}}(3)=2$, $children_{\mathcal{T}}(3)=\{x,4\}$, $leaves_{\mathcal{T}}(3)=\{x,y,z\}$, and $ancestor_{\mathcal{T}}(\{4,u\})=2$. Note that for any node N in $\mathcal{T}$, all nodes in $children_{\mathcal{T}}(N)$ share exactly the same path from the root of $\mathcal{T}$ to N and, thus, they all have the same shared metric value.

Fig. 4 gives the algorithmic details of the minimal tree construction technique. Intuitively, we use a bottom-up procedure to inspect internal nodes of the two topologies to determine whether they are equivalent, distinct, or inconsistent, building the minimally consistent tree as we go. For example, it is straightforward to verify that feeding tree of Figs. 3a and 3b to the `MinimalTree-Construction` function leads to the tree (Fig. 3d), while tree in Figs. 3a and 3c are inconsistent.

The construction uses $\mathcal{T}_2$ as a potential minimal tree, and is based on comparing each node N of $\mathcal{T}_1$ (in descending order of their depth) to a corresponding node W in $\mathcal{T}_2$, where $W = ancestor_{\mathcal{T}_2}(children_{\mathcal{T}_1}(N))$ to determine whether new nodes should be added to $\mathcal{T}_2$. The construction relies on keeping the following invariants: 1) By checking $\mathcal{T}_1$ internal nodes in descending order of their depth, the construction ensures that the subtrees rooted at an internal node being checked are consistent and minimal. 2) Whenever two nodes (one in $\mathcal{T}_1$ and the other in $\mathcal{T}_2$) have been assigned the same label N, then the rooted subtrees at these nodes are consistent and span the same set of clients $leaves_{\mathcal{T}_1}(N) = leaves_{\mathcal{T}_2}(N)$. The construction evaluates the nodes in $children_{\mathcal{T}_1}(N)$ and $children_{\mathcal{T}_2}(W)$ and considers four possible cases:

Case 1 (Exact Match): ($children_{\mathcal{T}_1}(N) = children_{\mathcal{T}_2}(W)$). In this base case, N and W are trivially consistent, and the construction simply assigns the label N to node W.

Case 2 (Reduction): ($children_{\mathcal{T}_1}(N) \subset children_{\mathcal{T}_2}(W)$). In this case, node N identifies greater sharing among nodes in $children_{\mathcal{T}_1}(N)$ than the level reflected by node W in $\mathcal{T}_2$. As a result, the construction inserts node N between W and $children_{\mathcal{T}_1}(N)$ in $\mathcal{T}_2$.

Case 3 (Compression): ($children_{\mathcal{T}_1}(N) \not\subseteq children_{\mathcal{T}_2}(W)$ AND $|leaves_{\mathcal{T}_1}(N)| = |leaves_{\mathcal{T}_2}(W)|$). In this case, node W reflects greater sharing than that of node N; and as a result, $\mathcal{T}_2$ already contains full information about this subtree and need not be updated.

Case 4 (Inconsistency): ($children_{\mathcal{T}_1}(N) \not\subseteq children_{\mathcal{T}_2}(W)$ AND $|leaves_{\mathcal{T}_1}(N)| \neq |leaves_{\mathcal{T}_2}(W)|$). In this case, nodes in the subtree rooted by node W represent an inconsistency as they should have appeared in the subtree rooted by node N in $\mathcal{T}_1$.

Construction Running Time: As shown in Fig. 4, the construction is a loop over all nodes in T_2 ($O(m)$ such nodes) and, for each of these nodes there is, in addition

```
MinimalTreeConstruction($\mathcal{T}_1, \mathcal{T}_2$)

for (all nodes $N$ in $\mathcal{T}_1$ in decreasing order of their depth) do
    let $W$ be the least common ancestor in $\mathcal{T}_2$ of the children of $N$ in $\mathcal{T}_1$

    if (children$_{\mathcal{T}_1}(N)$ == children$_{\mathcal{T}_2}(W)$) then      /* Exact match */
        Set the label of node $W$ in $\mathcal{T}_2$ to be $N$;

    else if (children$_{\mathcal{T}_1}(N)$ $\subseteq$children$_{\mathcal{T}_2}(W)$) then      /* Reduction */
        Insert new node $N$ in $\mathcal{T}_2$ with parent $W$ and children = children$_{\mathcal{T}_1}(N)$

    else if (leaves$_{\mathcal{T}1}(N)$ ==leaves$_{\mathcal{T}2}(W)$) then      /* Compression */
        Set the label of node $W$ in $\mathcal{T}_2$ to be $N$;

    else return NULL;      /*Inconsistent topology */

    return ($\mathcal{T}_2$);      /* $\mathcal{T}_2$ now is the minimal tree of $\mathcal{T}_1$ and old $\mathcal{T}_2$ */

endfor
```

Fig. 4. Minimal tree construction technique.

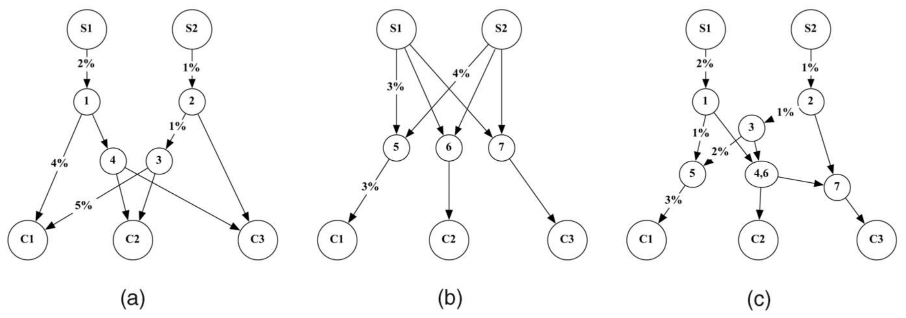

Fig. 5. Example of integrating MINT snapshots from servers to clients. Panel (a) depicts the integration of trees $\mathcal{T}_1$ and $\mathcal{T}_2$ from the server's perspective. Panel (b) depicts the integration of three trees from the client's perspective. Panel (c) depicts the integrated view incorporating all vantage points.

to a constant number of comparisons, a construction of $ancestor_{\mathcal{T}}(.)$, $leaves_{\mathcal{T}}(.)$, and $children_{\mathcal{T}}(.)$ which takes $O(m\log(m))$ time each. Also, there is a comparison between two $children_{\mathcal{T}}(.)$ sets which takes $O(m\log m)$ time. The overall running time of the construction is thus $O(m^2\log(m))$. This concludes the proof of the lemma and, as a consequence, the theorem. □

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 observations. 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 *servers. We say that these topologies are* mutually consistent *if there exists a graph $\mathcal{G}$ spanning all endpoints in $\bigcup \mathcal{T}_i$ and when $\mathcal{G}$ is restricted to the routing tree induced by shortest-path routing then $\mathcal{T}_i$ can be generated from that subgraph by repeated application of the collapse operation.*

Theorems 3 and 4 generalize the topology integration process to merge consistent topologies. Theorem 3 applies to topologies observed from an arbitrary set of servers to an arbitrary set of clients. Theorem 4 applies to topologies observed between an arbitrary set of collaborating servers.

Theorem 3. *Given a set of n servers $s_1, s_2, \ldots, s_n$, a set of m clients $c_1, c_2, \ldots, c_m$, and a procedure that enables the evaluation of:*

1. $f(p_{s_i\rightarrow c_k})\ \forall i = 1,2,\ldots,n$ *and* $\forall k = 1,2,\ldots,m$.
2. $f(p_{s_i\rightarrow c_k.s_i\rightarrow c_l})\ \forall i = 1,2,\ldots,n$ *and* $\forall k,l = 1,2,\ldots,m$, *where* $k \neq l$.
3. $f(p_{s_i\rightarrow c_k.s_j\rightarrow c_k}) \forall i,j = 1,2,\ldots,n$ *and* $\forall k = 1,2,\ldots,m$, *where* $i \neq j$.

For some monotone and separable metric f, one can efficiently infer *and* label *the base topology $\mathcal{G}$ induced by f over the physical topology induced through shortest-path routing and connecting the n servers to the m clients for any sensitivity parameter $c > 0$.*

Proof (Sketch). We only present the high-level steps in the proof as the main constructions follow directly from methods we have already derived. These steps are: 1) Inferring the metric induced trees $\mathcal{T}_i$ connecting each server s_i to the set of all clients c_k. By applying the construction provided in the proof of Theorem 1, $\mathcal{T}_i$ can be inferred and labeled. Fig. 5a shows the integration of $\mathcal{T}_1$ and $\mathcal{T}_2$ connecting each of two servers s_1 and s_2 to three clients c_1, c_2, and c_3. 2) Inferring the metric induced trees $\mathcal{B}_k$ connecting each client c_k to the set of all servers s_i. By applying the Corollary to Theorem 1, $\mathcal{B}_k$ can be inferred and labeled. Fig. 5b shows the integration of $\mathcal{B}_1, \mathcal{B}_2$, and $\mathcal{B}_3$ connecting the servers s_1 and s_2 to each of three clients c_1, c_2, and c_3. 3) Integrating the $\mathcal{T}_i$ and $\mathcal{B}_k$ trees to construct the MINT graph. Fig. 5c shows a sample graph resulting from the integration of the trees in Figs. 5a and 5b. The methods behind the construction leverage the ideas used in the proof of Theorem 1 in a straightforward manner. The algorithm is presented in Fig. 6 and a sample step of this construction is presented in Fig. 5.

Interpreting the algorithm in Fig. 6, we reconcile each triple of trees consisting of a pair of server trees $\mathcal{T}_i$ and $\mathcal{T}_j$, and a client tree $\mathcal{B}_k$. For triples of this form, we consider the paths $s_i \rightarrow c_k$ and $s_j \rightarrow c_k$ in $\mathcal{T}_i$, $\mathcal{T}_j$ and $\mathcal{B}_k$. Fig. 5a shows the paths from s_1 and s_2 to c_1 for $\mathcal{T}_1$ and $\mathcal{T}_2$; and Fig. 5b shows the paths from s_1 and s_2 to c_1 for $\mathcal{B}_1$ during one step of this merge process. □

Theorem 4. *Given a set of n servers $s_1, s_2, \ldots, s_n$ and a procedure that enables the evaluation of $f(p_{s_j})$, $f(p_{s_k})$, and $f(p_{s_j.s_k})$ for some monotone, separable and symmetric metric f and assuming symmetric routing, one can efficiently* infer *and* label *the base topology $\mathcal{G}$ induced by f over the physical topology induced through shortest-path routing and connecting the n servers for any sensitivity parameter $c > 0$.*

Proof (Sketch). The proof is based on the efficient inference of $\mathcal{T}_i$, connecting server i to the other $n-1$ servers, $\forall i = 1,2,\ldots,n$ the base topology induced by f over the physical topology connecting server s_i to the other $n-1$ servers (Theorem 1). Monotonicity and separability of f

```
MINTSnapshotsIntegration($\mathcal{T}, \mathcal{B}$)

let $\mathcal{G} = \mathcal{T}$
for (all pairs $\mathcal{T}_i$ and $\mathcal{T}_j$, $i \neq j$) do
    let $s_i$ and $s_j$ be the roots of $\mathcal{T}_i$ and $\mathcal{T}_j$, respectively
    for (all trees $\mathcal{B}_k$) do
        let $c_k$ be the root of $\mathcal{B}_k$
        let $S$ be the segment $p_{s_i \to c_k}.{}_{s_j \to c_k}$ in $\mathcal{B}_k$
        for (all identified nodes $x$ in $S$) do
            Insert a node $u$ in $p_{s_i \to c_k}$ in $\mathcal{G}$ such that $f(p_{s_i \to u})$ in $\mathcal{G}$ is equal to $f(p_{s_i \to x})$ in $\mathcal{B}_k$
            Insert a node $v$ in $p_{s_j \to c_k}$ in $\mathcal{G}$ such that $f(p_{s_j \to v})$ in $\mathcal{G}$ is equal to $f(p_{s_j \to x})$ in $\mathcal{B}_k$
            Merge nodes $u$ and $v$ and merge the paths $p_{u \to c_k}$ and $p_{v \to c_k}$ in $\mathcal{G}$
        endfor
    endfor
endfor
return $\mathcal{G}$
```

Fig. 6. Construction for integrating MINT snapshots from servers to clients.

ensure that we can infer and label $\mathcal{T}_i$. Also, symmetric routing ensures that a path connecting server s_i to server s_j is the same path connecting s_j to s_i in the reverse direction. This means that internal nodes on the path connecting s_i to s_j in $\mathcal{T}_i$ are also on the path connecting s_j to s_i in $\mathcal{T}_j$. It should be emphasized that symmetric routing is not widespread in current wide-area networks, such as today's Internet. Monotonicity and symmetry of f allow us to order the internal nodes on the path between s_i and s_j, and separability of f allows us to label every hop along this path. □

4 PERISCOPE IMPLEMENTATION

We have embodied our MINT framework in Linux by developing an API called PERISCOPE [20] (a Probing Engine for the Recovery of Internet Subgraphs) that implements the functionality necessary to infer and label metric-induced topologies, using the constructions presented in Section 3.[2] Currently, PERISCOPE is designed to infer MINT topologies by sending explicit probes (active probing), as opposed to doing the inference during the normal course of operation of existing applications. PERISCOPE server-side functionality consists of: 1) orchestrating probe transmission, 2) maintaining probe statistics relative to the metric of interest, and 3) running the inference and labeling processes. PERISCOPE requires *no* support from clients beyond the ability to respond to ICMP ECHO REQUESTs. Fig. 7 depicts the main components of the PERISCOPE architecture.

The `Manager` keeps a record of all endpoints (i.e., clients) under consideration. Endpoints are partitioned into application-defined *groups*—applications specify clients in a group through a custom system 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, whose implementation we discuss momentarily. 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 specific packet sequence to a subset of the group's endpoints, as required by the probing engine. The `Scheduler` ensures that the packets within a packet sequence are inserted in correct sequential order on top of the IP stack. For example, in order to send two packets back-to-back, we extended the kernel library with a dynamic, downloadable kernel module that sends two packets back-to-back with a single system call. The endpoints to which probes are targeted are selected by the `Scheduler` in a random fashion to avoid synchronization effects The `Monitor` keeps track of the probe 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 library. 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 periodic application callbacks to report inference and labeling results. This optimization is valuable for busy servers. In most cases, the procedure used to infer shared metric values requires sending *back-to-back* probe packets, which can only

2. The PERISCOPE API can be downloaded from http://www.csc.ncsu.edu/faculty/harfoush/periscope.

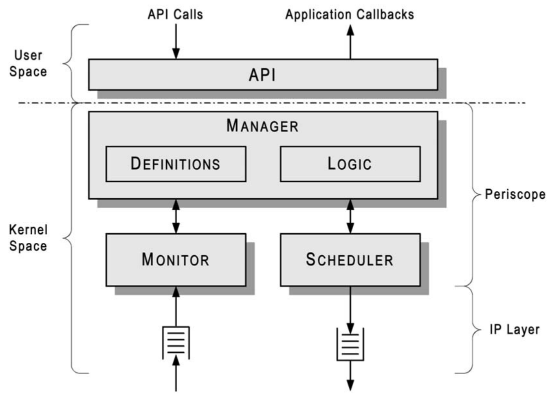

Fig. 7. PERISCOPE architecture.

be guaranteed by implementing the probing procedure in the kernel to avoid context-switching by the operating system. The interface between applications and PERISCOPE is done through the use of control sockets. System calls 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.

5 Case Study: Unicast Loss Topology Identification

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 accuracy of such a procedure as well as the extent to which such a labeling process scales must be analyzed in the context of the particular algorithm used. In this section, we focus our presentation on applying the theoretical results and the PERISCOPE embodiment presented earlier to a particular metric—namely, *loss rate*. Instantiations for other metrics such as queuing delay and capacity bandwidth are also possible using procedures such as those presented in [15], [21]. We use the term *loss topology* to refer to the instantiation of MINT for the loss rate metric. We begin this case study with a presentation of three criteria that enable us to evaluate the goodness of MINT inference and labeling. We then proceed to describe the specific procedure we used to measure shared losses between a sender and a pair of receivers. This procedure has also been imported into our PERISCOPE architecture. We then describe our experimental evaluation of our approach, conducted both by extensive *ns* [29] simulations and by validation through PERISCOPE Internet experiments.

5.1 Success Criteria for Inference and Labeling

We now introduce our three evaluation criteria. The first two criteria are used to evaluate the goodness of a MINT *inference* technique, whereas the third is used to evaluate the goodness of a MINT *labeling* technique. In practice, all three of these values will be computed by running experiments over a representative set of similar trees, computing the values 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 $t = 0$.

Definition 6. *The inference* accuracy $A(t)$ *of a MINT inference strategy at time t is the probability that the strategy yields the correct topology at time t.*

In our experiments, to measure accuracy at time t, we calculate the percentage of experiments in which the correct MINT topology was identified at time t. The accuracy criterion is absolute in the sense that it does not allow for a quantification of how "close" an inferred MINT topology is to the exact MINT topology, in the event that it is inaccurate. We will now introduce a discrepancy measure to provide such a relative quantification for tree topologies.

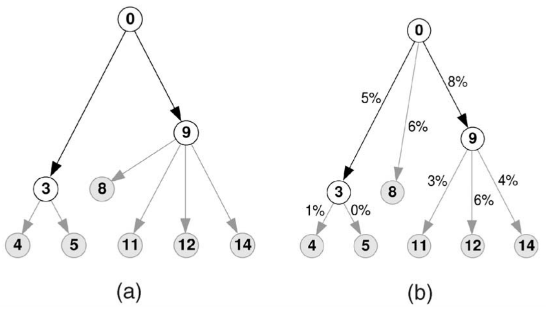

Fig. 8. Illustration of the use of the Inference Discrepancy and Labeling Error Measures. (a) Inferred tree. (b) Labeled tree.

Definition 7. *The inference* discrepancy $D(t)$ *of a MINT inference strategy on a tree topology T at time t is given by:*

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

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

To give an intuition for the discrepancy criterion, consider the loss topology shown in Fig. 1a and assume that as a result of applying an inference procedure with $c = 0.05$, the topology shown in Fig. 8a is obtained. The inferred topology is not identical to the correct $\mathcal{T}_{0.05}$ loss topology shown in Fig. 1d and the discrepancy between the inferred topology and $\mathcal{T}_{0.05}$ is $\sqrt{3/15} = 0.447$. Unlike quantification of the success of an inference process, which can be done independent of the metric(s) used to develop inferences, quantifying the success of a labeling process must necessarily reflect the metrical labels. The following definition applies specifically to labeled loss topologies.

Definition 8. *The labeling* error $E(t)$ *of a loss topology labeling process on a topology T at time t is*

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

where e_l denotes the correct loss probability for link l, $\hat{e}_l(t)$ denotes the measured loss probability for link l at time t, and L is the number of links in T.

To give an intuition for the labeling error measure, which can be interpreted as the average inaccuaracy in the labeling of an arbitrary link in a topology, consider the tree shown in Fig. 1a. Assume that the labeled topology shown in Fig. 8b is obtained. As a result of applying a labeling procedure with $c = 0.05$, obviously, the labels on that topology are not identical to the labels on the $\mathcal{T}_{0.05}$ loss topology shown in Fig. 1d. The labeling error is then

$$\sqrt{(0.02^2 + 0.01^2 + 0.01^2)/8} = 0.00866.$$

5.2 Bayesian Probing

Our procedure for identifying shared loss between a pair of clients is the Bayesian Probing (BP) technique developed in [19]. Consider clients 11 and 14 in the topology shown in Fig. 1a. 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 the portion that is not. 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 and PERISCOPE API. To that end, the technique uses two types of probe sequences:

Definition 9. *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 10. *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 intrapair packet spacing is at most ϵ and the interpair spacing is at least Δ time units.*

One-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 developed in [19] 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, provided that the queues are drop-tail gateways. (As noted in [19], the method is sensitive to the drop-tail assumption and is not effective when RED gateways are employed.) 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 near-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.

5.3 Performance Evaluation in `ns`

In this section, we present results of extensive `ns` [29] simulations that demonstrate the accuracy, convergence, and robustness of our approach.

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 bursty background traffic resulting from aggregating a set of Pareto ON/OFF UDP sources with a constant bit rate of 36Kbps during the ON times using 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 and labeling processes begin.

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

Fig. 9. Link congestion settings and resulting loss rates.

To represent the various levels of congestion that any of these links may exhibit, we have chosen three sets of parameters that reflect "High," "Mild," and "Low" levels of congestion. The baseline parameter settings for these congestion levels (and the resulting loss rates) are tabulated in Fig. 9. Our choice of very high loss rates (7-15 percent) for highly congested links was meant to stress-test our technique under severe congestion (we primarily observed 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 of $c = 4$ percent is small enough to observe links with losses > 4 percent. Our simulation results are slightly worse with such high 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 inference and labeling techniques, 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 five leaves were then selected as clients. The congestion level for each tree edge was then chosen at random from the link baseline models with the following distribution: 50 percent Low, 30 percent Mild, and 20 percent High.

Configuring Bayesian Probing: 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 our experiments, each probe sequence was generated using an independent Poisson process with a mean probing rate of 5 probes/sec, or 200ms average interpacket 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.

Experimental Setup: To determine the accuracy of our loss topology inference technique, we generated 20 5-client baseline trees at random, as described earlier. We ran the loss topology inference technique from the 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. The setup of the labeling experiment was similar, except that the `ns` server agent ran the labeling procedure on a provided (i.e., correct) loss topology. For each one of the 20 randomly generated trees, we ran the inference and labeling experiments 20 times, each time seeding the cross-traffic with a different random seed. The results reported below were then averaged over these 400 experiments.

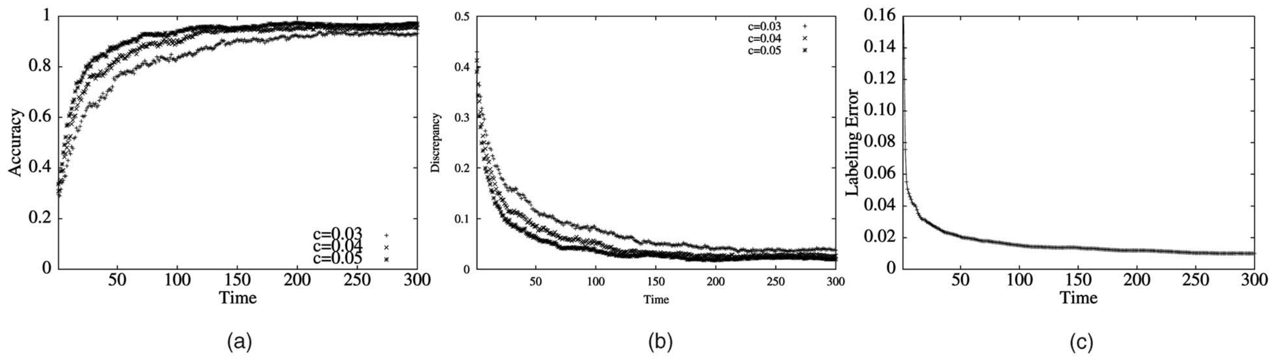

Fig. 10. (a) Accuracy, (b) discrepancy, and (c) labeling error of loss topology.

Results: Fig. 10 shows the accuracy and discrepancy 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. Fig. 10 indicates that both accuracy and discrepancy settle to within 10 percent of their steady-state values within 50 seconds. Fig. 10c shows the labeling error as a function of time. The labeling error converges to within 1 percent 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 two 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 loss on the second shared bottleneck loss is incorrectly assigned to independent links.

5.4 Internet Validation Using PERISCOPE

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.[3] Fig. 11 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 Fig. 1). Intermediate router IP addresses were obtained through the use of *traceroute*. The server is in the continental US 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 Fig. 11) is such a reference for $c = 0$. Obtaining such a reference for a nonzero sensitivity parameter is impossible since it requires knowledge of loss rates on all links of the logical tree. Moreover, 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 Fig. 11 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 Fig. 11. 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 four 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 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 interprobe independence assumption of the BP approach discussed in Section 5. Fig. 12a shows the percentage of PERISCOPE inferred trees that are found to be *inconsistent* with the logical tree in Fig. 11 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 nonstationarity 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 Fig. 11a. Indeed, PERISCOPE inferred six different loss topologies. Over the 20 experiments we conducted, the most frequently inferred loss

3. Validating our tool requires observing loss-topologies of appreciable structure—hence, our choice of an international set of endpoints. Our ongoing work on Internet loss topology characterizations to small sets of random endpoints (such as from CAIDA/NLANR logs) do not often yield rich and interesting structures. The depicted topology is meant to be illustrative, not representative. Also, experiments to the selected set of endpoints did not consistently reveal high losses. Fig. 11c was constructed only after integrating consistent inferred loss topologies viewed at different times.

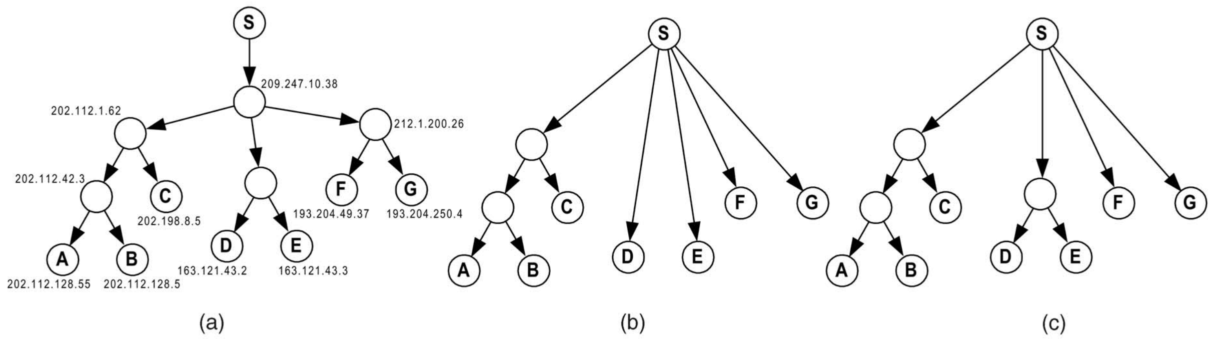

Fig. 11. PERISCOPE Validation: (a) Logical tree used as a test case, (b) most frequent inferred loss tree, and (c) minimal loss tree spanning all inferred trees.

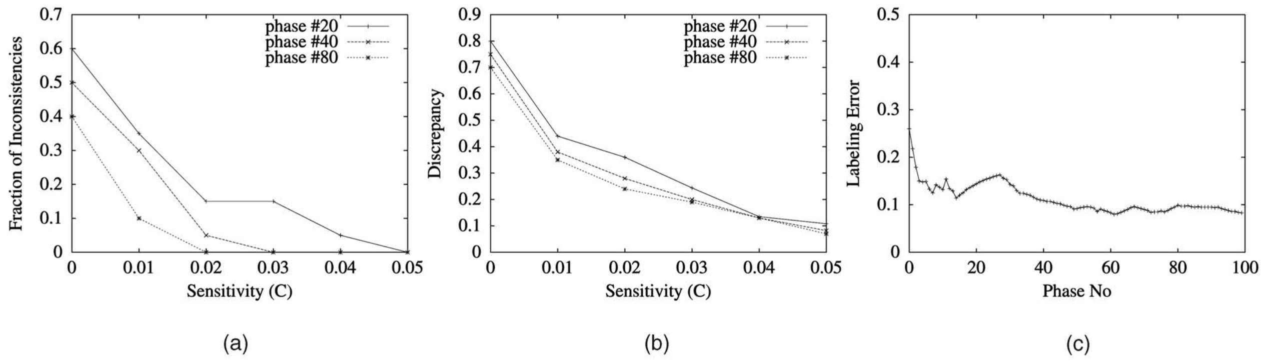

Fig. 12. PERISCOPE: (a) Inference inconsistency, (b) discrepancy, and (c) labeling error for the Internet topology shown in Fig. 11.

topology tree is shown in Fig. 11b. 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 Fig. 11c. Clearly, that tree is consistent with the logical tree depicted in Fig. 11a. To validate the labeling accuracy of PERISCOPE, we implemented a simple tool which repeatedly probes *all* nodes of the logical tree of Fig. 11, including internal nodes, concurrently with our use of PERISCOPE. Loss statistics obtained from this Poisson probing tool are compiled to yield the loss rates on the various links of Fig. 11. Then, 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 Fig. 12. Fig. 12b 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, Fig. 12c shows the reduction in the labeling error as the number of phases executed increases from 0 to 100 phases, spanning approximately 4 minutes.

6 Conclusion

In this paper, we presented MINT—a general framework for the representation of shared network resources as captured by a given metric of interest. One key contribution of our work is to classify metrics as satisfying monotonicity, separability, and symmetry properties, and to demonstrate the connections between these properties and our ability to infer and label MINT topologies using these metrics from one or more vantage points. Using the MINT framework, it is possible to create a "network caricature" highlighting only those network regions of interest, whether they be with respect to significant loss, delay, jitter, or other metric.

We then presented PERISCOPE, a publicly available embodiment of MINT constructions implemented in Linux. We instantiated MINT for a specific metric of interest—packet loss rates—and presented results of simulation and Internet measurement experiments that confirmed the effectiveness and robustness of our constructions over a wide range of network conditions.

Acknowledgments

This work was supported in part by US National Science Foundation grants ANI-9986397 and ANI-0093296. A preliminary version of results presented in this paper appeared in part in [18], [3], [20].

References

[1] A. Adams, T. Bu, R. Cáceres, N. Duffield, T. Friedman, J. Horowitz, F.L. Presti, S. Moon, V. Paxson, and D. Towsley, "The Use of End-to-End Multicast Measurements for Characterizing Internal Network Behavior," *IEEE Comm. Magazine,* May 2000.
[2] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," *Proc. ACM SIGCOMM,* 1999.

[3] A. Bestavros, J. Byers, and K. Harfoush, "Inference and Labeling of Metric-Induced Network Topologies," *Proc. IEEE Infocom,* June 2002.

[4] J.C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. ACM SIGCOMM,* pp. 289-298, Sept. 1993.

[5] T. Bu, N. Duffield, F. LoPresti, and D. Towsley, "Network Tomography on General Topology," *Proc. ACM SIGMETRICS,* June 2002.

[6] R. Cáceres, N.G. Duffield, J. Horowitz, D. Towsley, and T. Bu, "Multicast Based Inference of Network-Internal Characteristics: Accuracy of Packet-Loss Estimation," *Proc. IEEE Infocom,* Mar. 1999.

[7] R. Cáceres, N.G. Duffield, S.B. Moon, and D. Towsley, "Inference of Internal Loss Rates in the MBone," *Proc. IEEE Global Internet (Globecom) Conf.,* 1999.

[8] R. Caceres, N. Duffield, and T. Friedman, "Impromptu Measurement Infrastructures Using RTP," *Proc. IEEE Infocom,* June 2002.

[9] R. Carter and M.E. Crovella, "Measuring Bottleneck Link Speed in Packet Switched Networks," *Proc. PERFORMANCE '96, Int'l Conf. Performance Theory, Measurement and Evaluation of Computer and Comm. Systems,* Oct. 1996.

[10] M. Coates and R. Nowak, "Network Loss Inference Using Unicast End-to-End Measurement," *Proc. ITC Conf. IP Traffic, Modeling and Management,* Sept. 2000.

[11] N. Duffield, J. Horowitz, F. LoPresti, and D. Towsley, "Multicast Topology Inference from Measured End-to-End Loss," *IEEE Trans. Information Theory,* vol. 48, no. 1, Jan. 2002.

[12] N. Duffield, J. Horowitz, D. Towsley, W. Wei, and T. Friedman, "Multicast-Based Loss Inference with Missing Data," *IEEE J. Selected Areas of Comm.,* vol. 20, no. 4, May 2002.

[13] N. Duffield, F.L. Presti, V. Paxson, and D. Towsley, "Inferring Link Loss Using Striped Unicast Probes," *Proc. IEEE Infocom,* Apr. 2001.

[14] N. Duffield, J. Horowitz, and F.L. Presti, "Adaptive Multicast Topology Inference," *Proc. IEEE Infocom,* Apr. 2001.

[15] N. Duffield, J. Horowitz, F.L. Presti, and D. Towsley, "Network Delay Tomography from End-to-End Unicast Measurements," *Proc. Int'l Workshop Digital Comm.,* Sept. 2001.

[16] R. Govindan and A. Reddy, "An Analysis of Internet Inter-Domain Routing and Route Stability," *Proc. IEEE Infocom,* Apr. 1997.

[17] T. Griffin and G. Wilfong, "An Analysis of BGP Convergence Properties," *Proc. ACM SIGCOMM,* pp. 277-288, Sept. 1999.

[18] K. Harfoush, "A Framework and Toolkit for the Effective Measurement and Representation of Internet Internal Characteristics," PhD dissertation, Boston Univ., Aug. 2002.

[19] K. Harfoush, A. Bestavros, and J. Byers, "Robust Identification of Shared Losses Using End-to-End Unicast Probes," *Proc. Eighth IEEE Int'l Conf. Network Protocols (ICNP),* Nov. 2000.

[20] "`PeriScope`: An Active Probing API," *Proc. 2002 Passive and Active Measurement Workshop, PAM '02,* Mar. 2002.

[21] "Measuring Bottleneck Bandwidth of Targeted Path Segments," *Proc. IEEE Infocom,* Apr. 2003.

[22] "IPMA: Internet Performance Measurement and Analysis," http://www.merit.edu/ipma, 2005.

[23] V. Jacobson, "`Pathchar`: A Tool to Infer Characteristics of Internet Paths," ftp://ftp.ee.lbl.gov/pathchar, 2005.

[24] S. Keshav, "Congestion Control in Computer Networks," PhD dissertation, Univ. of California at Berkeley, Sept. 1991.

[25] K. Lai and M. Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay," *Proc. ACM SIGCOMM,* Aug. 2000.

[26] "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth," *Proc. USENIX Symp. Internet Technologies and Systems,* Mar. 2001.

[27] F. LoPresti, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-Based Inference of Network-Internal Delay Distributions," *IEEE/ACM Trans. Networking,* vol. 10, no. 6, Dec. 2002.

[28] "Mtrace: Tracing Multicast Path between a Source and a Receiver," ftp://ftp.parc.xerox.com/pub/netsearch/ipmulti, 2005.

[29] "`ns`: Network Simulator," http://www.isi.edu/nsnam/ns/, 2005.

[30] V. Padmanabhan, "Addressing the Challenges of Web Data Transport," PhD dissertation, Univ. of California at Berkeley, Sept. 1998.

[31] J.-J. Pansiot and D. Grad, "On Routes and Multicast Trees in the Internet," *Computer Comm. Rev.,* vol. 28, no. 1, pp. 41-50, Jan. 1998.

[32] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," PhD dissertation, UC Berkeley and Lawrence Berkeley Laboratory, 1997.

[33] S. Ratnasamy and S. McCanne, "Inference of Multicast Routing Trees and Bottleneck Bandwidths Using End-to-End Measurements," *Proc. IEEE Infocom,* pp. 353-360, Mar. 1999.

[34] D. Rubenstein, J. Kurose, and D. Towsley, "Detecting Shared Congestion of Flows via End-to-End Measurement," *IEEE/ACM Trans. Networking,* vol. 10, no. 3, June 2002.

[35] S. Seshan, M. Stemm, and R. Katz, "SPAND: Shared Passive Network Performance Discovery," *Proc. Usenix Symp. Internet Technologies and Systems (USITS),* Dec. 1997.

[36] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and Modeling of the Temporal Dependence in Packet Loss," *Proc. IEEE Infocom,* pp. 345-352, Mar. 1999.

Azer Bestavros received the SM degree in 1988 and the PhD degree in 1992, both in computer science from Harvard University. He is currently a professor and chairman of computer science at Boston University. Professor Bestavros' research interests are in the general areas of networking and real-time systems. His seminal works include his generalization of the classical rate-monotonic analysis to accommodate probabilistic guarantees, his pioneering of the push model for Internet content distribution adopted years later by CDNs, his characterization of Web traffic self-similarity and reference locality, his development of various caching and streaming media delivery protocols, and his development of efficient techniques for inference of network caricatures using real-time end-to-end measurement. He received distinguished service awards from both the IEEE and the ACM. He served as chair, officer, or PC member of most major conferences in real-time and networking systems, including ICNP, Infocom, Sigmetrics, Sigmod, RTSS, RTAS, and ICDE. His research has been funded by grants totaling more than $12M from various government agencies and industrial labs. He is a member of the IEEE and the IEEE Computer Society.

John W. Byers is an assistant professor of computer science at Boston University. Prior to joining BU, he received the PhD degree in computer science at the University of California at Berkeley in 1997 and was a postdoctoral researcher at the International Computer Science Institute in Berkeley in 1998. His research interests include algorithmic aspects of networking, Internet content delivery, and network measurement. Dr. Byers received a US National Science Foundation CAREER Award in 2001, and the IEEE ICDE Best Paper award in 2004. He serves on the program committees for numerous conferences, including ACM SIGCOMM, ACM SIGMETRICS and IEEE Infocom. He is currently on the editorial board of *IEEE/ACM Transactions on Networking*, and has been a member of the ACM since 1999.

Khaled A. Harfoush received the PhD degree in computer science from Boston University in 2002. He is currently an assistant professor in the Department of Computer Science at North Carolina State University, which he joined in 2002. His research interests are in the general areas of network modeling, Internet measurement, peer-to-peer systems, and network security. Professor Harfoush is a recipient of the prestigious US National Science Foundation CAREER award. He serves on the program committees for numerous conferences including IEEE INFOCOM and IEEE ICNP. He is a member of the IEEE and the IEEE Computer Society.