Advanced Indexing Techniques for Wide-Area Network Monitoring

Xin Li[†], Fang Bian[†], Hui Zhang[†],
Christophe Diot[*], Ramesh Govindan[†], Wei Hong[‡], Gianluca Iannaccone[*]

***Abstract*— Detecting and unraveling incipient coordinated attacks on Internet resources requires a distributed network monitoring infrastructure. Such an infrastructure will have two logically distinct elements: distributed monitors that continuously collect packet and flow-level information, and a distributed query system that allows network operators to efficiently and rapidly access this information. We argue that, in addition to supporting other types of queries, the network monitoring query system must support multi-dimensional range queries on traffic records (flows, or aggregated flow records). We discuss the design of MIND, a distributed indexing system which supports the creation of multiple distributed indices that use proximal hashing to scalably respond to range queries.**

I. Introduction

Numerous network monitoring systems have been designed in the recent past for operational and research uses. The motivation for such systems includes performance monitoring, security enforcement (anomaly and intrusion detection, DoS attack detection *etc.*), and network troubleshooting and maintenance [5][6][8][11][23]. However, such systems are often designed to monitor specific subsystems and are mostly centralized in design.

A natural next step is the design of distributed monitoring systems. Such systems consist of two logically distinct components: a distributed network traffic monitoring system, and a distributed querying system. The traffic monitoring system consists of passive monitors that collect and store (perhaps for a limited time) detailed network traces [14]. An analogy has been made [18] that these monitors collectively form a sensor network.

These monitors can, in addition, generate traffic summaries in the form of flow records or suitably aggregated and filtered versions thereof. We envision the distributed (for scale and robustness reasons) querying system as allowing users to efficiently query these traffic summaries. Such a capability can allow network operators to examine the performance of flow aggregates (*e.g.,* traffic from or to a customer), correlate flows to detect anomalies or intrusions *etc.* More importantly, this capability can indicate precisely which network monitors contain relevant traces that can be further analyzed (to confirm a potential intrusion, for example), *i.e.,* such a system lets users efficiently drill down to important data locations.

In this paper, we examine the interplay between networking and databases in designing the querying system. Specifically, while we envision this distributed query system to consist of several components (*e.g.,* a scalable DHT system that allows exact-match queries) we focus on one kind of functionality crucial to network monitoring: support for multi-dimensional range queries. Many queries on traffic summaries are naturally expressed as multi-dimensional range queries (*e.g.,* was there a flow of size greater than 1MB to customer prefix P in time interval T).

We consider the motivation for, and sketch the design of, MIND [1], a system that supports the creation of distributed multi-dimensional indices. MIND consists of a collection of network nodes that forms a hypercube overlay; these nodes are logically distinct from, but could be co-located with, network monitors. Traffic summaries, expressed as multi-attribute tuples and generated at network monitors, can be inserted into one or more indices. MIND routes these tuples to nodes such that tuples near each other in the attribute space are likely to be stored at the same node, making multi-dimensional range searches efficient.

Care must be taken in using a distributed index for network monitoring. Clearly, it is not feasible to insert all flow records from each network monitor into MIND; such an approach could incur significant traffic overhead. Rather, we see MIND as being used in much the same way database administrators build centralized indices. A network administrator performs careful off-line analysis to decide the attributes to be indexed and the granularity of traffic summaries to be inserted into MIND. This database design analysis is based on the trade-off between the cost of building the index, and the expected frequency of querying the system. We analyze the feasibility of using MIND for detecting suspicious flows and analyzing network performance. We conclude with a summary of the research challenges arising in its design.

II. Motivation

While most existing network monitoring systems are centralized[2], research is just beginning to explore the design of distributed network monitoring systems. Such systems consist of two logically distinct components: a distributed traffic monitoring system and a distributed query system.

The traffic monitoring system is composed of a number of passive traffic monitors deployed around the network. Generally, a traffic monitor collects and stores packet traces (either the entire sequence of packets observed at each link

[†]Computer Science Department, University of Southern California. Email: {xinli, bian, huizhang, ramesh}@usc.edu
[*]Intel Research at Cambridge, UK. Email: {christophe.diot, gianluca.iannaccone}@intel.com
[‡]Intel Research at Berkeley, USA. Email: wei.hong@intel.com

[1]Multi-dimensional Indices for Network Diagnosis

[2]In this paper, we focus on traffic monitoring. Existing systems sometimes monitor the control plane (*e.g.,* routing traffic) as well. Our arguments apply equally to such systems.

attached to the monitor, or a sampled version of that sequence). The monitor may be capable of storing a large amount of traffic observed at a site to allow off-line detailed analysis of network data. In addition, each traffic monitor can generate traffic summaries, called flow records, in real-time. A flow record typically counts the volume of traffic, optionally over a specific time window, of a traffic aggregate. Traffic aggregates can be flexibly defined in terms of nodes, applications, or collections thereof. Thus, a traffic aggregate defined by the destination http://www.usc.edu and a port number (say, 80) measures the volume of Web traffic to that destination. Another example of a traffic aggregate is the Web requests sent from a set of nodes to another set of nodes. Often, given the way node addresses are assigned, "interesting" sets of nodes are usually represented by IP address prefixes (*e.g.,* 128.9.160/20). In database terms, flow records can be defined by a view over raw packet traces typically involving windowed aggregates and various predicates over IP addresses. The design of traffic monitors represents a significant ongoing research challenge [6][14][21].

Existing centralized network monitoring systems typically collect and store flow records at a single site. For flows at the granularity of individual TCP connections, centralized network monitoring doesn't scale well. Data from the Abilene backbone [1] indicates flow rates from single routers in excess of 900 flows per second, and a single large ISP might wish to instrument several hundred links. Finally, a centralized store of flow records is a single point of failure and can become a query hotspot. For all these reasons, in this paper we consider a distributed monitoring system.

Prior work [13][26] has discussed the design of distributed approaches to evaluating certain kinds of declarative queries over traffic aggregates. One kind of query that naturally arises in this context is the *multi-dimensional range query*. A flow record can be [19][7] represented by a tuple in a multi-dimensional attribute space. The dimensions in this space include the source and destination IP addresses, the source and destination port numbers, and possibly time. Thus, we argue, a distributed system that supports multi-dimensional range queries should be an essential component of future wide-area network monitoring systems.

The design of distributed systems for supporting range queries has started to receive some attention in the literature [20][4][25][3]. Many of these systems store multi-dimensional data in a manner that preserves *locality* — data tuples are routed to nodes such that tuples stored at a node are "nearby" in the attribute space. This locality-preserving hashing enables efficient querying, since queries can also be routed (using the same mechanisms as insertions) to nodes that contain the relevant parts of the attribute space. Existing techniques that support range queries cannot be directly applied to network monitoring.For example, several of these systems [25], [3], [12] cannot be easily extended to support multi-dimensional range queries. Mercury [4] replicates data records to an extent that may be incompatible with the volume of flow records. In DIM [20], the logical routing structure is closely tied to geography, and the system itself is designed for a different set of constraints. However, DIM's design has inspired many of the mechanisms in MIND.

This paper makes two contributions. First, it discusses the design of a software infrastructure, MIND, that supports the dynamic creation of distributed multi-dimensional indices. The design of MIND is informed by the characteristics of the network monitoring application. Second, it argues that network monitoring needs are best served by carefully creating appropriate indices based on the network monitoring task (*e.g.,* detecting suspicious flows) rather than, for example, creating a single large index of flow records. The former approach is akin to the traditional database design problem, where indices are created based on careful analysis of query patterns and data distributions. We substantiate the feasibility of this approach by analyzing data from two large Internet backbones. The following sections describe these two contributions.

III. MIND: A Component of the Query System

The logical structure of an N-node MIND is an N-node hypercube. Nodes join MIND in a manner similar to that proposed in [2] to ensure a balanced hypercube where the maximum degree of any node is minimized. MIND assigns each node a code which naturally corresponds to its location on the hypercube, as shown in Figure 1.

Now consider a multi-dimensional attribute space. To embed this data space into the hypercube overlay, MIND partitions the data space into smaller hyper-rectangles by recursively sub-dividing each attribute in cyclic order, until the number of hyper-rectangles equals the number of nodes in the hypercube. Now, each hyper-rectangle (or *zone*) can be assigned a code based on this division procedure. Figure 2 shows a 3-dimensional data space that has been divided to 8 zones, each with a unique code, according to Figure 1. (In our example, each attribute space is normalized to a real number between 0 and 1, but our approach generalizes to discrete spaces as well.) MIND assigns each zone to the node with the matching code. This mapping is *locality-preserving* in the sense that data within the same zone are mapped to the same node and data in neighboring zones are mapped to neighboring nodes on the overlay.

MIND indices are implicitly created by inserting a tuple; each inserted tuple is self-identifying, in that it contains a globally unique name for the index, together with a compact description of the schema for the index. Tuples may be inserted into MIND from any node in the network. That node first computes the prefix of the tuple's zone code as follows. If the length of node A is k, then node A needs to encode tuple T to at most k bits in order to see if tuple T should be stored at A or not. For example, if the code of tuple T starts with `1` while node A's code starts with `0`, then the node A stops encoding T right after the first bit and simply delivers T to one of its neighbors whose code starts with `1`. Note that on a hypercube, such a neighbor must exist.

Routing a tuple in MIND is fairly straightforward: code-based greedy routing. However, MIND routing has an interesting feature: message destinations are refined step-wise, and each node makes routing decisions independently. In general, in a N-node MIND, each node only needs to know $\log N$ data

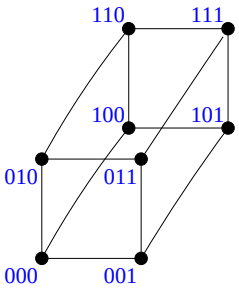

Fig. 1: The hypercube structure of a 8-node MIND network.

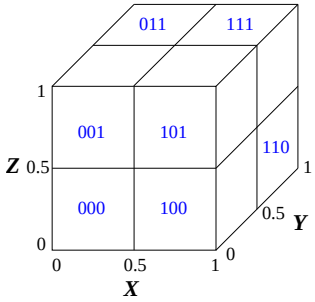

Fig. 2: The mapping from a 3-dimensional data space to the MIND network show in Figure 1.

space partition points in order to route messages correctly. Trivially, the average routing path length on the hypercube overlay is $O(\log N)$, which is also the average insertion cost of MIND indices.

A multi-dimensional range query in MIND defines a hyper-rectangle in the data space and in general, intersects more than one zone. The code of a range query is defined as the code of the common prefix of all zones intersecting the query. Thus, queries can be routed to these nodes using an algorithm similar to insertions. The response to a range query is all the tuples contained in the hyper-rectangle. Query processing in MIND is done by continuous and distributed splitting of a large range query into smaller subqueries such that ultimately, each resulting subquery is completely contained in a zone and so can be resolved at a single node. If, for any reason, the response to a subquery is lost, this can easily be detected at the query originator since it manifests itself as a "hole" in the query hyper-rectangle. Duplicate responses can also be detected in the same way.

The query cost depends on the size of a query with respect to the number of nodes covered by the query. In general, the average query cost depends on the average query size, *i.e.,* the average number of nodes a query needs to visit. For queries that follow exponential or Zipfian distributions, for example, we expect the average query cost will be dominated by $O(\log N)$ for large N.

IV. Using MIND for Network Monitoring

In this section we describe three sample queries that illustrate how network monitoring applications might use multi-dimensional indices. For each query, we describe the monitoring objective, the format of the tuples that need to be inserted in the index and the English or SQL description for the query. In all the examples, it may be desirable to index any subset of attributes depending on the query workload. We point out what we believe are the common cases.

A. Finding Suspicious Flows

Given the complexity of inspecting packet streams at line rates, one would like to spot "suspicious flows" first and then run complex anomaly detection algorithms only on a subset of identified suspicious flows. A common application of finding suspicious flows is to find machines that run port scans (for gathering information or just because it is a worm). An important measure is short-connection attempts without data exchange, that would result in a large number of connections from a given source (we call this number the fanout of a source). The generic query for suspicious flows would be:

> Q_1: *Find all the sources that attempted to connect to more than F hosts in destination prefix(es) D.*

Note that destination D (as well as source S if used) is in fact an IP prefix and so represents continuous ranges of IP addresses. Q_1 defines a 5-dimensional data space T_1:

$$T_1\colon \langle destination, fanout, source, node, timestamp\rangle.$$

In most cases, one will build an index on the first three attributes. As an example, we translate Q_1 to a generic SQL statement as follows.

SELECT * FROM T_1
WHERE $destination = D$ AND $fanout \geq F$.

More specific variants of Q_1 can be derived from the above SQL statement, *e.g.,* using *node* in the select list or specifying time durations in the WHERE clause.

B. Performance Analysis

Given a network customer (*i.e.,* an IP prefix), one would like to find TCP connections to or from that customer that have exchanged a number of packets (or bytes) large enough to allow to estimate classic performance metric such as round-trip times and loss rates [28][15]. The generic query for performance analysis can be stated as:

> Q_2: *Find all flows destined for D (and/or sourced from S) that have carried at least O octets (or in between O_1 and O_2).*

Q_2 also defines a 5-dimensional data space T_2:

$$T_2\colon \langle destination, source, octets, node, timestamp\rangle.$$

An index on the first three attributes will be the usual choice.

C. Finding Camouflaged Applications

Finally, this query is used to identify network applications that are using well-known ports to work around firewalls and packet filters (e.g., peer to peer applications [16]) or that tunnel their traffic using other application layer protocols to avoid connection charges (e.g., DNS tunneling [22]). The queries for camouflage detection will ask for the connections that have carried a large number of packets/bytes (called flow size) just for, say a DNS/ICMP request, or the connections that have transferred a disproportionate amount of HTTP traffic to the same IP addresses. These IP addresses would potentially be

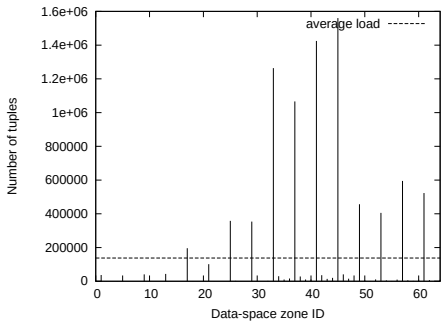

Fig. 3: The tuple distribution from one-day real-world network traffic traces.

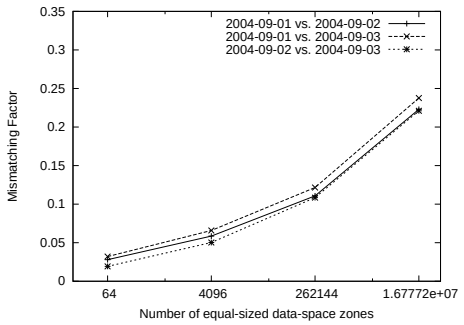

Fig. 4: The daily difference of tuple distributions of our real-world network traffic traces.

the other sides of tunneling. More specifically, we have the following generic query:

> Q_3: *Find all the flows either from source S, or to destination D, or both, that have flow size more than X and/or with destination port P.*

Q_3 defines a 6-dimensional data space T_3:

$$T_3: \langle source, destination, flow\text{-}size, port, node, timestamp \rangle$$

In most cases, indexing on the first 3 attributes is sufficient, especially when the inserted tuples are selected (filtered) based on port numbers of interest (e.g., port 80 for HTTP, 53 for DNS).

V. Research Challenges and Future Work

Some of the performance issues of MIND can be naturally addressed with previously proposed techniques. For example, proximity routing can be achieved with the scheme in [27]. In this section, we address some of the research challenges that lie ahead in the design and development of MIND, namely: load balancing, robustness, and query optimization.

A. Load Balancing

In a system such as MIND, one can talk about two kinds of load balancing: balancing routing load, and balancing storage. Routing load balancing can be achieved by maintaining a balanced hypercube, using replication to alleviate query hot-spots.

Storage load balancing presents instead additional challenges. Remember the mapping from the data space to nodes in MIND is locality-preserving. When the data distribution is skewed, a careful partition is needed for balanced storage. We analyzed the traffic data collected from two large networks, Abilene [1] and Geant [10], to understand what data distribution a MIND system could expect. The network data consists of Cisco Netflow [5] records continuously gathered from all the routers in the Abilene and Geant network. Netflow records contain detailed information on the flows that traverse each router, such as destination and source IP addresses and port numbers, byte and packet counts, time of the first and last packet of the flow, *etc.*. We collected these tuples and aggregated them according to the fields indicated in the three indices defined in Section IV.

Figure 3 shows the tuple distribution of the sum of the three indices over one day of traffic traces. The x-axis represents 64 equal-sized zones (boxes) which compose the whole data space, and the y-axis depicts the number of tuples falling onto each of the zones. Clearly, even with a coarse granularity of 64, we see that the data is significantly skewed.

While other systems [17][9][4] have considered migrating data from one node to another, that approach entails significant volumes of data movement in MIND. Rather, in MIND, we periodically (on, say, a diurnal time scale) adjust the mapping from the data space to the overlay space (*i.e.,* re-compute the hyper-rectangles associated with each node) based on the data distribution. This leverages the observation that network traffic, in the aggregate, does not change dramatically from day to day.

To validate this observation we have analyzed the collected daily real-world network traffic data. We use *mismatching factor* to measure the drifting of the indexed data in the attribute space from day to day. The attribute space is partitioned into K equal-sized zones. For zone x ($1 \leq x \leq K$), we define

$$L_i(x) = \frac{\text{number of Day-i records fallen into x}}{\text{average number of Day-i records per zone}}.$$

The mismatching factor between the data distributions of Day-i and Day-j, $MF(i,j)$, is then defined as

$$MF(i,j) = \sum_{i=1}^{K} \frac{|L_i(x) - L_j(x)|}{2}.$$

Intuitively, $MF(i,j)$ describes the total data in percentage that needs to be moved for the transform between the data distributions of Day-i and Day-j. Furthermore, $MF(i,j)$ will become larger when K gets bigger. Figure 4 shows that the difference of the data distributions between adjacent days can be as low as 0.2, which justified that the tuple distribution is stable and change slowly over days [3]. This clearly suggests that it is feasible to daily remap the data space onto MIND's overlay.

While a centralized solution for histogram generation is affordable in MIND, we are investigating a de-centralized approach to minimize communication overhead.

B. Robustness

The main robustness concern here is to provide high data availability through replication. The choice of replication sites should be considered under two conditions:

[3]Notice that in a data set with steady daily distribution, the hourly/minutely data distributions may still be highly dynamic. The locally sensitive *node-migration* approach could cause excessive and unnecessary data transferring under this situation.

1) A replica should not be too "far" away from the original data; otherwise, data replication will bring a high overhead.
2) A replica should be easy to retrieve if the original data is not available.

There is a rather straightforward and efficient choice for replication in MIND: the *sibling node*. The sibling of a node A, $sib(A)$ is defined to be the node that has exactly the same code as A except the last bit. If such as node does not exist, due to node join and leave, then the $sib(A)$ is the node that shares the longest common prefix with A. Whenever a tuple is inserted in A, a replica of the tuple will be delivered to and stored at $sib(A)$. When node A fails, in order to maintain a balanced hypercube, we adopt the node leave handling proposed in [2]. Whichever node will take the space left by node A, it can easily retrieve the data from $sib(A)$.

C. Data Base Design and Query Optimization

We point out, without detailed analysis, that the following condition must be satisfied in order for MIND indices to be more efficient than simple flooding for range queries that follow Zipfian or exponential distributions: $\frac{Q}{I} > \frac{\log N}{N}$, where Q stands for the total number of queries, I for the total number of insertions, and N for the total number of nodes.

As in traditional databases, the efficient way to use MIND is to create indices based on query workload. For example, if for canonical query Q_2, the majority of the query predicates are on attributes *destination* and *octets*, then building an index on attributes *destination*, *source*, and *octets* will be less efficient than an index on only *destination* and *octets*.

The complexity comes when there are more than one dominating query forms. For example, suppose that a user have only two types of queries: q_1 on *octets* and q_2 on *destination* and *source*. If the probability of q_1 is much less than that of q_2, a single index on all the three attributes will be sufficient. However, if the probability of q_1 equals that of q_2, we need to consider to build two separate indices one for q_1 and the other for q_2, and treat both as secondary indices containing partial tuples with unique tuple IDs.

When multiple indices are used in answering the same query, MIND needs support *join* operations among indices. We can create MIND indices for distributed rendezvous for performing joins similar to [13] using DHTs. We can also leverage the histograms that we collect for load balancing to intelligently choose an efficient join order between indices. We plan to experiment with different distributed query execution and optimization techniques.

VI. Conclusion

In this paper, we have discussed the query processing component of a generic network monitoring architecture, namely MIND, a multidimensional range query support system for wide-area networks. While the fundamental design draws from traditional database indices and DHT research, the special requirements of high speed network environments have placed non-trivial research challenges. We are currently exploring the design of MIND through implementation and testing on the PlanetLab [24].

References

[1] Abilene Backbone Network. http://abilene.internet2.edu.
[2] M. Adler, E. Halperin, R. M. Karp, and V. V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proc. of STOC'03*, pages 575–584.
[3] J. Aspnes and G. Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, Baltimore, MD, USA, 12–14 Jan. 2003.
[4] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of SIGCOMM 2004*.
[5] Cisco Systems. NetFlow services and applications. White Paper, 2000.
[6] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of ACM Sigmod*, June 2003.
[7] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of SIGCOMM 2003*, pages 137–148.
[8] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, R. Rockell, D. Moll, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 2003.
[9] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer to Peer Systems. In *Proc. of VLDB 2004*, Toronto, Canada.
[10] GEANT Backbone Network. http://www.dante.net.
[11] M. Grossglauser and J. Rexford. Passive traffic measurement for IP operations. *to appear in The Internet as a Large-Scale Complex System, Oxford Press*, 2005.
[12] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties, Mar. 2003.
[13] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of VLDB*, 2003.
[14] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The CoMo White Paper. Technical Report IRC-TR-04-017, Intel Research, Sept. 2004.
[15] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proceedings of IEEE Infocom*, Mar. 2004.
[16] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of P2P traffic. In *Proceedings of ACM Sigcomm Internet Measurement Conference*, Oct. 2004.
[17] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of SPAA 2004*, pages 36–43.
[18] J. Kurose. Networking: successes, new challenges, and an expanding waist as the field approaches 40. Infocom 2004 keynote talk, March 2004.
[19] T. V. Lakshman and D. Stiliadis. High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching. In *Proc. of SIGCOMM 1998*, pages 203–214.
[20] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the ACM Sensys*, Nov. 2003.
[21] A. Moore, J. Hall, E. Harris, C. Kreibech, and I. Pratt. Architecture of a network monitor. In *Proceedings of Passive and Active Measurement Workshop*, Apr. 2003.
[22] Name Server Transport Protocol. http://www.freshmeat.net/projects/nstx.
[23] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23), Dec. 1999.
[24] PlanetLab Network. http://www.planet-lab.org/.
[25] S. Ramabhadran, J. M. Hellerstein, S. Ratnasamy, and S. Shenker. Prefix hash tree: An indexing data structure over distributed hash tables, 2004.
[26] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems. In *Proceedings of the Second Workshop on Hot Topics in Networking*, Nov. 2003.
[27] H. Zhang, A. Goel, and R. Govindan. Incrementally improving lookup latency in distributed hash table systems. In *Proc. of SIGMETRICS, 2003*.
[28] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *Proceedings of ACM Sigcomm*, Aug. 2002.