CoMo: An Open Infrastructure for Network Monitoring – Research Agenda

Gianluca Iannaccone
Intel Research Cambridge

February 17, 2005

1 Introduction

The CoMo project will build an open infrastructure for network monitoring. The infrastructure will span multiple administrative domains and allow its users to gather data following per-system data sharing policies. Users will be able to query the CoMo systems to perform any processing on the network traffic.

Given the open nature of the infrastructure the main challenge is then

"how to efficiently handle multiple, competing, traffic queries?"

Our idea is to define a per-query accuracy metric and maximize the aggregate accuracy of all query results given the available resources. We argue that in network monitoring it is always possible to regulate the accuracy of a query (that will include the measurement error as well as the timeliness of the results). For a given set of queries we then need to define an aggregate utility function and be able to exercise a fine-grained control over it.

To overcome this challenge we need to address two orthogonal *sub-challenges*. The first has to do with allowing users to post *any* query to the systems in a *secure* fashion. The second is about the *definition and regulation* of queries' accuracy in a *distributed* monitoring infrastructure. We now describe the two sub-challenges in detail and introduce our ideas to overcome them.

#1 *"how to allow users to easily define and compute any query on the network traffic while providing security guarantees to the network operators sharing the data?"*

This challenge can be addressed at two levels. First, at the architectural level where we define the boundaries and the capabilities of the system. Then, with a feature-rich query language that allows an efficient specification and implementation of any traffic metric and at the same time makes possible to statically verify the "safety" of the implementation. As a side note, it is not an aim of CoMo to perform distributed query processing, instead we anticipate that other applications will gather data from multiple CoMo systems and fuse them

to perform the processing (e.g., IrisNet [18]) We will discuss the CoMo system architecture in Section 2. The security guarantees will be discussed in Section 3.

#2 *"how to deliver sufficiently timely and accurate answers to a query when there may be insufficient processing resources to give complete results?"*

The CoMo systems will make policy decisions to turn down incoming queries or reduce the accuracy of running queries in order to satisfy the requests of "most" queries. The design space for addressing this challenge can be defined in two dimensions: in space, i.e., if the policy is a global decision applied to all systems or a local decision that each system may take independently; in time, i.e., if the policy is applied online at query run-time, or off-line during the query planning phase. Sections 4, 5 and 6 present three solutions within this design space.

2 Architecture

Joint work with C. Diot, D. McAuley (IRC), A. Moore, I. Pratt (U. Cambridge) and L. Rizzo (U. Pisa). Please refer to [24] for details.

In the design of the CoMo architecture we pursue three objectives: (*i*) provide data protection for the network operator; (*ii*) support any traffic query; (*iii*) handle a wide range of operating environments (different transport media, different input data rates, different query loads).

In [24] we present the details of the CoMo architecture that aims to satisfy the above objectives. Here we highlight the main ideas.

Data protection. A major obstacle to the deployment of a monitoring infrastructure across multiple administrative domains is the anxiety of network operators when facing a request to share network information with other parties.

The solution that is often used is to share an anonymized version of the data and allow users to *own* the anonymized data. This approach is known to have severe limitations [34]. First, it does not really address the concern of the network operators that is never convinced of having anonymized "enough". As a result, today, no commercial ISP or corporation is sharing any anonymized packet traces. Moreover, the user is also not satisfied because the anonymization process discards too much information rendering the data useless for many applications.

The approach we pursue in CoMo is instead to *move the queries to the data*. The operator maintains the ownership of the data while users send queries where the data reside. A network operator may then inspect the query beforehand and decide whether that query should be allowed to run on the dataset. We envision network operators to define the policy for the kind of queries depending on the user's role. For example, users belonging to the same organization may be allowed to post any queries while external users may be restricted to a short list of allowed queries.

Support any traffic query. The CoMo infrastructure must allow users to compute any metric on the traffic. The first question is: *"how can a user express the metric to compute?"*.

The need for expressing any traffic metric forces us to discard commonly used query language (e.g., SQL) because too restrictive (how to express a query to perform automated worm signature detection [29] or to track TCP sender state [25]?). Furthermore, we need to pick a language that is of common use among network researchers and application developers.

To address this challenge, we choose a classical modular approach that has proven to be successful in similar contexts [36, 6]. Users write a single (simple) plug-in module in C, making use of a feature-rich API provided by the core system of CoMo. Then, they can send the module source code directly to the system(s) to "load" the query and run it on the traffic.

The next question is then: *"how does the system find the data of interest for the metric?"*. Supporting any queries means supporting queries that need to process packets as they arrive to the interface but also packets that have been transmitted in the past. There is a growing interest in network forensic applications, i.e., applications that can look back in the traffic to understand and trace back the root cause and behaviors of specific network events [23, 35]. For this reason, we envision the CoMo system to constantly maintain a large packet trace always available to queries. The length of the packet trace would obviously depend on disk space and traffic rate but we aim to dimension the system to maintain 72 hours of traffic at least (i.e., one entire week-end).

Finally, the third question is *"what should our target query response time be?"*. Unfortunately, different queries have a wide range of response time requirements. In the context of capacity planning, the matter is the accuracy of the answer rather than the response time [38]. For traffic engineering, common tools and techniques work in a time-frame of tens of minutes given that that is the timescale at which BGP operates [40]. Anomaly or worm detection applications may request (near) real-time responses [35, 41, 29]. Clearly, real-time queries cannot be answered fetching packets from disk. For this reason, our architecture provides two data feeds for queries. Real-time queries run directly on incoming packets while queries with less stringent response time requirements may run at a later time fetching packets from disk.

Handle different operating environments. The architecture guarantees the scalability of the system by carefully partitioning the tasks among the different system components. Queries reside in plug-in modules. A set of *core processes* makes sure that modules can process the packets, maintain state information about the packets, store data to disk and send the query response back to the user. Core processes are partitioned according to the requirements of the functionalities they provide, e.g., functions with real-time requirements are run in different processes from where functions with less stringent requirements are executed. Moreover, each hardware device is assigned to a different process. This partitioning allows to efficiently and easily scale the system up to higher speeds and larger query loads.

3 Safe Modules

Before discussing the security of the system we have to define the CoMo threat model. We assume the following: (*i*) we trust the hardware, the operating system and the core processes[1]; (*ii*) an adversary can assume the credentials of any valid user in the system, write any plug-in module and load the module in any CoMo system.

The security guarantees we aim to provide are the following:

- A malicious module cannot starve any other modules of resources. This problem can be reduced to a classical resource management problem and solved by guaranteeing fair access to system resources of all modules. In fact, because modules process incoming traffic, on which we have little if any control, even a perfectly legal module could be driven to monopolize all processing resources in response to certain input patterns.

- A malicious module cannot corrupt input data of other modules. This is achieved by isolating modules and removing any dependency between them. In CoMo, the input data of a module are provided by the core processes, never by plug-in modules loaded by other users. This design choice negatively impacts the performance of the overall system because modules cannot be pipelined to perform complex functions (i.e., the same computation may have to be performed by multiple modules). However, it has several advantages: (*i*) the input data of all modules is pre-defined; (*ii*) the input data of a module cannot be corrupted by other users; (*iii*) it allows to treat each module independently, to measure its resource usage in isolation and thus implement resource management decisions more efficiently.

- A malicious module cannot gain access to the state of other modules or to packet fields that system policies do not allow to read. This results in controlling memory accesses of all modules. One approach could be to provide memory isolation via virtualization (e.g., running modules as separate processes). This approach has the drawback of delegating the control over module resource usage to the OS or, if modification to the OS are needed to gain more control, would reduce the portability of the CoMo software requiring to run part of the core system in kernel space.

  Instead, we prefer to investigate the use of safe languages where memory access are guaranteed to happen only within defined regions of memory. CoMo modules are defined as a set of callback functions that operate in three memory regions: packet data (read only), packet meta-data (read/write but private to the module) and a third memory region shared among all instances of the same module.

  We aim to define a C-like language for modules adapting existing solutions proposed in the literature [27, 14].

[1] Although expanding the threat model to include these system components would raise several interesting research questions we consider them as out of scope for this project.

4 Resource management via packet sampling

Joint work with C. Barakat (INRIA) and C. Diot (IRC). For more details please refer to [2].

Traffic sampling refers to the action of collecting a subset of the traffic on a network link. At a high level, sampling is concerned with making partial observations on the traffic, from which conclusions can be drawn on properties that apply to all packets. The *observation problem* is concerned with minimising information loss whilst reducing the volume of collected data as much as possible. It is this reduction which makes the collection process scalable. The way in which the partial information is transformed into knowledge of the packet stream as a whole is the *inversion problem*. The inversion process is in general imperfect and introduces errors.

Sampling is therefore an efficient way to reduce the load on a traffic monitor. It allows to perform data reduction close to the wire with little impact on the processing resources of the monitoring system.

In the context of the CoMo project, sampling may allow a graceful degradation of the performance of a running module. Sampling the incoming traffic of a running module is therefore a *local, on-line* policy decision that a CoMo system may take to reduce overall processing resource usage, while keeping the results of the module within the user-requested error bounds.

Several works in the literature have studied the inversion problem from sampled traffic. Duffield et al. [12] study the problem of flow splitting and propose estimators for the total number of flows and for the average flow size in the original traffic stream. [13, 21] study the inversion of the flow size distribution with two different methods. They both show that the major difficulty comes from the number of flows that are not sampled at all and that need to be estimated with an auxiliary method. [8] finds the sampling rate that assures a bounded error on the estimation of the size of flows contributing to more than some predefined percentage of the traffic volume. [11] introduces the idea of smart sampling where the purpose is to isolate flows that contribute considerably to the traffic; this is done by selecting flow records with a probability that increases with the flow size.

All prior work focused on the inversion of aggregate flow properties. However, it is not clear how packet sampling affects the accuracy of traffic measurement that focus on individual flow properties. As a first step in this direction, we model and analyze how to *detect and rank the largest flows from the sampled traffic*.

We choose to start from this metric because the knowledge of the top users or applications is one of the most useful statistics to be extracted from network traffic. This information is used for marketing purposes by application developers or content providers. Network operators use the knowledge on the most popular destinations to identify emerging markets and applications or to locate where to setup new Points of Presence. Content delivery networks use the popularity to define their caching or replication strategies. In traffic engineering, the identification of heavy hitters in the network can be used to treat and route them differently across the network [40, 37, 16]. Keeping track of the network prefixes that generate most traffic is also of great importance for

anomaly detection mechanisms and systems. A variation in the pattern of the most common applications may be used as a warning sign and trigger careful inspection of the packet streams.

Given all these potential applications, it does not come as a surprise that there has been a significant effort in the research community to find ways to track frequent items in a data stream [9, 10, 19, 7]. This problem has usually been addressed with the objective of reducing storage space usage. All the works in the literature assume that if the algorithm and the memory size are well chosen, the largest flows can be detected and ranked with a high precision. However, in the presence of packet sampling, even if the methods rank correctly the set of sampled flows, there is no guarantee that the sampled rank corresponds to the original rank.

We define the problem as follows. Consider a monitor that, for a given measurement period, samples packets independently of each other with probability p (random sampling) and classifies them into flows ("sampled flows"). At the end of the measurement period, the monitor sorts all sampled flows, based on their sampled size in packets, and returns an ordered list of the t largest flows. Because of the random nature of sampling, this sampled list may not match the list that could have been obtained without sampling. We try to answer the two following questions: (i) do the top t flows in the original unsampled traffic appear in the list (*detection*)? (ii) do they appear in the list in the correct order (*total ranking*)?

To answer the two questions, we perform an analytical study of the problem of ranking the sampled flows and compute the probability that they are *misranked*. Surprisingly, our analytical analysis indicates that a high sampling rate is required to obtain a good ranking. For example, considering a link with thousands of flows with a Pareto flow size distribution, the sampling rate must be above 10% to correctly rank the largest flows. We find that a sampling rate of 1% – as reccomended my most router vendors to avoid overloading the routers – allows only the successful ranking of the largest few flows, unless the number of flows on the monitored link is in the order of millions. We find that a coarser flow definition (e.g., packets destined to the same IP routable prefixes) improves the ranking accuracy only if the relative sizes of the largest flows increase as a function of the square root of their sizes. Therefore, contrary to common belief, having larger flow sizes does not always help in accurately detecting and ranking the largest flows.

Future research directions include devising additional methods to refine the ranking in presence of errors. We identify two possible directions: protocol-aware ranking and distributed sampling.

Protocol-aware ranking makes use of specific information carried in some of the packets (e.g., TCP sequence numbers) to refine the ranking of the largest flows. Preliminary investigations show that this approach is very promising. The main limitation is that it can only be applied to a subset of flows and flow definitions (e.g., TCP connections) and, depending on information set by end-hosts, may incur in errors in the presence of malicious hosts.

Distributed sampling tries instead to correlate sampled information from

multiple sites in the case they observe the same flow. This way we artificially increase the sampling rate for some flows and thus can report a more accurate estimate of their size. On the other hand, this approach introduces a bias in the ranking process since the sampling error strongly depends on the number of times the same flow is sampled.

5 Placement of CoMo systems and modules

Joint Work with C. Barakat (INRIA), G. Cantieni, P. Thiran (EPFL), C. Diot (IRC). Please refer to [5] for more details.

The CoMo infrastructure incurs in three types of costs: (*i*) the one-time deployment cost that refers to the actual cost of the hardware equipment (e.g., network capture device, splitter, disks, etc.); (*ii*) an operating recurring cost for running and maintaining the systems; (*iii*) a service cost that is the cost of running one specific query on the systems given the limited processing resources.

In order to minimize the deployment costs, network operators need to identify a set of limited *strategic locations* in their networks. The strategic importance of a location may depend on properties of the network (e.g., links connecting sites with a large number of users), of the links (e.g., transatlantic links, peering links), of the locations (e.g., available rack space), or it may be motivated by the demands and preferences of the users of the infrastructure.

Identifying the strategic locations is a hard problem and the placement of the monitoring devices is closely related to the measurement objective. For example, in [26], the authors focus on the placement of measurement devices for active monitoring (more specifically for the construction of distance maps). Bejerano et al. [3] address the placement problem for an active monitoring infrastructure to measure delays and detects faults. In [42], the authors address the network coverage problem, i.e., define a set of location that allow to observe most network traffic. They show that the problem of finding the optimal placement is NP-hard and present a set of greedy heuristics that provide near-optimal solutions.

In the context of CoMo, placing system addresses only half of the problem. It remains to be defined on how many and what systems a given query should run to maximize the accuracy while keeping the service cost to a minimum. We seek a method that given a query would allow to specify in a *global, off-line* fashion the target systems where the query should run and the sampling rate for the incoming traffic.

We define the problem as follows: given a network topology, a set of origin-destination pairs[2], and a monitoring "budget", which links should we monitor, and at what sampling rate to maximize the accuracy metric while keeping the monitoring cost below the total budget?

Note that we do not explicitly define the accuracy metric given that we aim to apply this same method to different queries. In this framework, we can define an optimization problem with non-linear constraints and derive an optimal solution as long as the accuracy metric is a strictly concave function of

[2] A pair may refer to nodes in the network, links, network prefixes or communicating end-points.

the sampling rate. We are currently investigating how to derive the behavior of the accuracy metric for some canonical network traffic queries.

Future work will also address a set of additional concerns from an operation standpoint related to the placement problem:

Resiliency to traffic changes. Traffic demand variations are expected in large networks given the constant evolution of the network topology with the addition of new nodes and users and the introduction of different traffic engineering practices.

Resiliency to network failures. The placement should exhibit little sensitivity to routing matrix changes. It has been shown that link failures are part of everyday operations but the vast majority of the failures are isolated, short-lived and the network returns to its pre-failure state in a matter of minutes [33]. These network failures cause the routing matrix to be very dynamic.

Competing traffic engineering objectives. Standard traffic engineering practices may also compete with the measurement goal and hamper the ability of the operator to optimize the monitoring infrastructure. For example, the practice of traffic load balancing across equal cost paths makes the placement problem more difficult to solve. Indeed, the knowledge of the traffic matrix and the routing matrix does not help in determining the actual path packets will take across the network. This is due to the randomness introduced at each hop by the load balancing strategy.

6 Distributed indices

Joint work with F. Bian, R. Govindan, X. Li, H. Zhang (USC), C. Diot (IRC) and W. Hong (IRB). For more details please refer to [31].

A CoMo system allows detailed analysis of past network data ("network forensic analysis"). Given the large amount of data stored in the system, it is desirable to be able to reduce the amount of data a query needs to process to produce the results. A natural approach is to build indices for the stored data.

Moving from a single system to a network of systems, the next step is the design of distributed indexing systems. The CoMo system can generate traffic summaries (in the form of flow records, for example) and allow users to efficiently query this set of traffic summaries in order to find which monitors contain relevant traces that can be further analyzed, allowing users to *drill down* to important data locations.

Prior work [22, 43] has discussed the design of distributed approaches to evaluating certain kinds of declarative queries over traffic aggregates. One kind of query that, surprisingly, has not received much attention is the *multi-dimensional range query*. A flow record is naturally [30, 15] represented by a hyper-rectangle 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. Many queries on network traffic are therefore 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). Thus, we argue, a distributed system that supports multi-dimensional range queries

should be an essential component of CoMo.

The design of distributed systems for supporting range queries has started to receive some attention in the literature [32, 4, 39, 1]. 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 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 based on locality-preserving hashing cannot be directly applied to network monitoring: some are designed for constrained environments [32], others do not support multi-dimensional range queries [1, 20], and yet others replicate data records to an extent that may be incompatible with the volume of flow records [4].

We consider the motivation for, and sketch the design of, MIND (Multi-dimensional Indices for Network Diagnosis), 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, the CoMo systems. Traffic summaries expressed as multi-attribute *tuples* 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.

Some of the research challenges that lie ahead in the design and development of MIND are the following:

Load Balancing Load balancing lies in two levels: routing (i.e.,, how to reach the node that contains the part of the index) and storage (i.e.,, where the index should be stored over time). Routing load balancing can be achieved by maintaining a balanced hypercube, while alleviating query hot-spots with replication. 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. Prior works [4, 28, 17] addressed this problem in one-dimensional data space with the method of *node migration*: light-loaded nodes leave their positions on the overlay space to re-partition the overlay space assigned for heavy-loaded nodes. While this approach can handle load dynamics well, it has three drawbacks in the context of MIND: (i) Node leave and rejoin will cause a skewed

hypercube structure. (*ii*) The huge amount of tuples in MIND makes node migration prohibitive. (*iii*) Local volatility will bring network oscillation such that nodes are repeatedly moving back and forth, never converging to a stable states. This is especially true when time-stamp is put into the data space for indexing.

There are logically two layers in the mapping: a mapping from the data space to the overlay space, and a mapping from the overlay space to nodes. *Node migration* addresses load balancing by adjusting the latter. We believe that in MIND adjusting the former is a better choice as it does not affect the hypercube structure. We are investigating a de-centralized approach to remap the data space.

Robustness MIND is a task specialized system and its nodes are not expected as dynamic and various as those assumed in DHTs. The main robustness concern here is to provide high data availability through replication. The choice of replication sites should be considered under two conditions: 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.

Query Optimization As in traditional databases, the efficient way to use MIND is to build *query-aware* indices. For example, if the majority of the queries 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 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 use classical secondary indices techniques such as partial tuples with universal id numbers to deal with it.

When multiple indices coexist in MIND, it will be necessary to support *join* across indices. We consider this to be out of scope for the CoMo project and plan to leverage on existing solutions or current on-going research projects within Intel that focus explicitly on distributed query processing.

7 Conclusion

We have presented here the main activities that form the research agenda of the CoMo project. There is another large set of activities that are part of the CoMo project and, although not discussed here, are instrumental to its success. They include performance optimization techniques (e.g., load balancing, how to efficiently monitor resource usage, how to predict query response times, etc.), ease of use and deployment (e.g., how to make the development of new modules

simple and efficient, what libraries and services the CoMo core systems should provide) and, finally, developing modules to use CoMo.

Acknowledgments

This document is the result of a large number of useful discussions with many people involved at various levels in the CoMo project: Simon Crosby, Carl Dellar, Christophe Diot, Rob Ennals, Joe Hellerstein, Larry Huston, Brad Karp, Lukas Kencl, Derek McAuley, Andrew Moore, Ian Pratt, Luigi Rizzo, Timothy Roscoe, Richard Sharp, Nina Taft.

References

[1] J. Aspnes and G. Shah. Skip graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, Jan. 2003.

[2] C. Barakat, G. Iannaccone, and C. Diot. Ranking flows from sampled traffic. Technical report, Intel Research, Feb. 2005.

[3] Y. Bejerano and R. Rastogi. Robust monitoring of link delays and faults in IP networks. In *Proceedings of IEEE Infocom*, Apr. 2003.

[4] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of ACM Sigcomm*, Sept. 2004.

[5] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the monitor placement problem: Optimal network-wide sampling. Technical report, Intel Research, Feb. 2005.

[6] D. Carney et al. Monitoring streams - a new class of data management applications. In *Proceedings of VLDB*, 2002.

[7] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of ICALP*, 2002.

[8] B. Y. Choi, J. Park, and Z. Zhang. Adaptive packet sampling for flow volume measurement. Technical Report TR-02-040, University of Minnesota, 2002.

[9] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of ACM PODS*, June 2003.

[10] E. Demaine, A. Lopez-Ortiz, and I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of 10th Annual European Symposium on Algorithms*, 2002.

[11] N. G. Duffield and C. Lund. Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure. In *Proceedings of ACM Sigcomm Internet Measurement Conference*, Oct. 2003.

[12] N. G. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *Proceedings of ACM Sigcomm Internet Measurement Workshop*, Nov. 2002.

[13] N. G. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of ACM Sigcomm*, Aug. 2003.

[14] R. Ennals, R. Sharp, and A. Mycroft. Linear types for packet processing. In *Proceedings of European Symposium on Programming*, 2004.

[15] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM Sigcomm*, Aug. 2003.

[16] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM Sigcomm*, Aug. 2002.

[17] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer to peer systems. In *Proceedings of VLDB*, Aug. 2004.

[18] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing*, 2(4), Oct. 2003.

[19] P. B. Gibbons and Y. Matias. Synopsis structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.

[20] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of Usenix*, Mar. 2003.

[21] N. Hohn and D. Veitch. Inverting sampled traffic. In *Proceedings of ACM Sigcomm Internet Measurement Conference*, Oct. 2003.

[22] 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.

[23] A. Hussain, J. Heidemann, and C. Papadopoulos. Identification of repeated dos attacks using network traffic forensics. Technical report, USC, July 2004.

[24] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The CoMo white paper. Technical report, Intel Research, Sept. 2004.

[25] 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.

[26] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *Proceedings of IEEE Infocom*, Apr. 2000.

[27] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. In *Proceedings of Usenix Conference*, June 2002.

[28] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of SPAA 2004*, 2004.

[29] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th Usenix Security Symposium (Security 2004)*, Aug. 2004.

[30] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of ACM Sigcomm*, Aug. 1998.

[31] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone. Advanced indexing techniques for wide-area network monitoring. In *Proceedings of NetDB Workshop*, Apr. 2005.

[32] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of ACM Sensys*, Nov. 2003.

[33] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *Proceedings of IEEE Infocom*, Mar. 2004.

[34] J. Mogul. Trace anonymization misses the point. World Wide Web Conference, July 2002.

[35] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of IEEE Infocom*, Mar. 2003.

[36] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The Click modular router. In *Proceedings of ACM Symposium on Operating Systems Principles*, Dec. 1999.

[37] K. Papagiannaki, N. Taft, and C. Diot. Impact of flow dynamics on traffic engineering design principles. In *Proceedings of IEEE Infocom*, Hong Kong, China, Mar. 2004.

[38] K. Papagiannaki, N. Taft, Z. li Zhang, and C. Diot. Long-term forecasting of internet backbone traffic: Observations and initial models. In *Proceedings of IEEE Infocom*, San Francisco, USA, Mar. 2003.

[39] S. Ramabhadran, J. M. Hellerstein, S. Ratnasamy, and S. Shenker. Prefix Hash Tree: An indexing data structure over distributed hash tables. Technical report, Intel Research, 2004.

[40] A. Shaikh, J. Rexford, and K. G. Shin. Load-sensitive routing of long-lived IP flows. In *Proceedings of ACM Sigcomm*, Sept. 1999.

[41] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *Proceedings of WORM*, Oct. 2004.

[42] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating network monitors: Complexity, heuristics and coverage. In *Proceedings of IEEE Infocom*, Mar. 2005.

[43] 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.