Provenance-aware Declarative Secure Networks

Wenchao Zhou Eric Cronin Boon Thau Loo
University of Pennsylvania

Abstract

In recent years, network accountability and forensic analysis have become increasingly important, as a means of performing network diagnostics, identifying malicious nodes, enforcing trust management policies, and imposing diverse billing over the Internet. This has lead to a series of work to provide better network support for accountability, and efficient mechanisms to trace packets and information flows through the Internet. In this paper, we make the following contributions. First, we show that network accountability and forensic analysis can be posed generally as data provenance computations and queries over distributed streams. In particular, one can utilize provenance-aware declarative networks with appropriate security extensions to provide a flexible declarative framework for specifying, analyzing and auditing networks. Second, we propose a taxonomy of data provenance along multiple axes, and show that they map naturally to different use cases in networks. Third, we suggest techniques to efficiently compute and store network provenance, and provide an initial performance evaluation on the P2 declarative networking system with modifications to support provenance and authenticated communication.

1 Introduction

The Internet was not designed with accountability as its primary goal. However, network accountability and forensic analysis have become increasingly important in recent years, as a means of performing network diagnostics, identifying malicious and misbehaving users, enforcing trust management policies, and imposing diverse billing over the Internet. This has lead to a series of proposals (e.g. [18, 3, 19, 10, 21, 4, 8, 11]) on improving network support for accountability, and efficient mechanisms to trace packets and information flows through the Internet. While there have not been a lack of proposals, several of them narrowly tackling a specific functionality or network application.

Provenance (also called *lineage*) has been studied in many different contexts. In the context of database systems, they have primarily used in databases to help "explain" to users why a tuple exists [6]. In this paper, we show that network accountability and forensic analysis can be posed generally as data provenance computations and queries over distributed streams. We argue that declarative networks [12, 14, 13] enhanced with the ability to maintain provenance of computations will enable a general extensible framework for specifying, analyzing, and auditing networks. Declarative networks utilize a database query language for specifying and implementing networks, and its dataflow framework captures information flow naturally as distributed streams computations. We further demonstrate that with the appropriate security extensions [1] to the query language used in declarative networks, we can further allow provenance computations and queries to be authenticated in untrusted environments.

Contributions and Organization: In Section 2, we provide a background on declarative networks, its query languages, and recent security extensions obtained by unifying its core language with logic-based access control languages. Next, in Section 3, we survey various use cases of network provenance ranging from real-diagnostics, forensics, accountability, and trust management. In Section 4, we then provide a taxonomy of different types of data provenance (local vs distributed, online vs offline, authenticated, etc), several of which are derived from existing database literature, and show that they map naturally into existing use cases. We outline some possible optimizations (Section 5), perform initial performance evaluations based on extensions to the P2 declarative networking system (Section 6), and then conclude in Section 7.

2 Declarative Networks

As background, we briefly introduce declarative networking and its query language, including security extensions. The high level goal of *declarative networks* [14, 13, 12] is to build extensible network architectures that achieve a good balance of flexibility, performance and safety. Declarative networks are specified using *Network Datalog* (*NDlog*), which is a distributed recursive query language used for querying network graphs. *NDlog* queries are executed using a distributed query processor to implement the network protocols, and continuously maintained as distributed views over existing network and host state.

Declarative queries such as *NDlog* are a natural and compact way to implement a variety of routing protocols and overlay networks. For example, traditional routing protocols can be expressed in a few lines of code [14], and the Chord [20] distributed hash table in 47 lines of code [13]. When compiled and executed, these declarative networks perform efficiently relative to imperative implementations.

2.1 Network Datalog Language

NDlog is based on Datalog [16]: a Datalog program consists of a set of declarative *rules*. Each rule has the form `p :- q1, q2, ..., qn.`, which can be read informally as "`q1` and `q2` and `...` and `qn` implies `p`". Here, `p` is the *head* of the rule, and `q1`, `q2`,...,`qn` is a list of *literals* that constitutes the *body* of the rule. Literals are either *predicates* with *attributes* (which are bound to variables or constants by the query), or boolean expressions that involve function symbols (including arithmetic) applied to attributes. (Predicates in datalog are typically relations, although in some cases they may represent functions.)

Datalog rules can refer to one another in a cyclic fashion to express recursion. The order in which the rules are presented in a program is semantically immaterial; likewise, the order predicates appear in a rule is not semantically meaningful. Commas are interpreted as logical conjunctions (*AND*). The names of predicates, function symbols, and constants begin with a lowercase letter, while variable names begin with an uppercase letter. We illustrate *NDlog* using a simple example of two rules that computes all pairs of reachable nodes:

```
r1 reachable(@S,D) :- link(@S,D).
r2 reachable(@S,D) :- link(@S,Z), reachable(@Z,D).
```

The rules `r1` and `r2` specify a distributed transitive closure computation, where rule `r1` computes all pairs of nodes reachable within a single hop from all input links (denoted by the `neighbor`, and rule `r2` expresses that "if there is a link from `S` to `Z`, and `Z` can reach `D`, then `S` can reach `D`." By modifying this simple example, we can construct more complex routing protocols, such as the distance vector and path vector routing protocols.

NDlog supports a *location specifier* in each predicate, expressed with `@` symbol followed by an attribute. This attribute is used to denote the source location of each corresponding tuple. For example, all `reachable` and `link` tuples are stored based on the `@S` address field. The output of interest is the set of all `reachable(@S,D)` tuples, representing reachable pairs of nodes from `S` to `D`.

When executed, the above *NDlog* query is essentially a distributed stream computation, where stream of `neighbor` and `reachable` tuples are joined at different nodes to compute routing tables. Interestingly, sliding windows commonly used in stream processing enables the *soft-state* [17] maintenance of network data: the window size essentially corresponds to the lifetime of all routes.

2.2 Secure Network Datalog

Secure Network Datalog (*SeNDlog*) [1] is a unified declarative language for networks and security policies. It combines language features from *NDlog*, and *Binder*, a logic-based language for access control in distributed systems, and *NDlog*, with the goal of providing a unified declarative language for networks and security policies. *SeNDlog* utilizes Binder's notion of *context* that represents a component (or security principal) in a distributed environment and a distinguished operator "`says`". We illustrate *SeNDlog* via the same reachable example as before, with the additional use of the "`says`" operator:

```
At S:
s1 reachable(S,D) :- link(S,D).
s2 linkD(D,S)@D :- link(S,D).
s3 reachable(Z,Y)@Z :- Z says linkD(Z,S),
                       W says reachable(S,Y).
```

The rules `s1-s3` are within the context of the principal `S`. An additional *localization rewrite* [12] ensures that all rule bodies are localized within a context (i.e. have the same location specifier). Assuming an untrusted network, this allows rules to execute only based on trusted local data, or authenticated data from remote sources. The "`says`" construct is an abstraction for the details of authentication. In one specific implementation, communication happens via signed certificates, where derived tuples signed using the private key of the exporting context can be imported into another context and checked using the corresponding public key. E.g. node `S` will import the `reachable(S,Y)` fact from its neighbor `W`, and verify that it is indeed from `W` via the signature stored with the fact. Node `S` then derives the `reachable(Z,Y)` fact which is signed and exported to node `Z`.

Note that the implementation of "`says`" may depend on the system and its context. In a hostile world, "`says`" may require digital signatures, while in a more benign world, "`says`" may simply append a cleartext principal header to a message—and this will of course be cheaper.

3 Provenance in Practice

In this section, we survey a (non-exhaustive) list of existing work in the networking literature that motivates the use of network provenance. We classify the use-cases as *real-time diagnostics*, *forensics*, *accountability*, and *trust management*. While our examples focuses on examples at the IP layer, we note that network provenance similarly applies to overlay networks, and other multi-hop networks such as sensor networks.

3.1 Real-time Diagnostics

Provenance (also called lineage) has been studied in many different contexts, but primarily to help "explain" to users why a tuple exists [6]. This is particularly useful to perform real-time diagnostics. One can add declarative rules that monitor a protocol for run-time anomalies, e.g. lack of convergence, network traffic spike suggesting possible intrusion etc. To illustrate, the following rule raises an alarm when the current best path from `S` to `D` via the next hop `Z` is greater than 10:

```
routeAlarm(@S,D,Z,C) :- nextHop(@S,D,Z,C), C>10.
```

When `routeAlarm` is generated, one can further execute a distributed query over the provenance of `nextHop` to figure out how the errant `nextHop` was computed. An alternative is to track changes of `nextHop` over past T seconds, and raise an alarm when the number of changes exceed a threshold, as an indication of divergence. For real-time diagnostics, typically, one only need to maintain the provenance for existing network state that is necessary to determine how an routing entry was generated.

3.2 Forensics

Real-time diagnostics involve running queries over current network state. In many cases, historical data is required in order to correlate traffic patterns of attackers. A common area of research has been in providing "traceback"[18] of traffic, either by the receiver or by an involved third party(e.g. in transitor networks), to determine where packets are originated from without trusting the unauthenticated IP headers.

We illustrate the relationship between traceback and provenance via the following *NDlog* example. Consider an IP router node `Node`, with forwarding table `nextHop(@Node, Dest, NextHop, Cost)`, and an incoming `packet(@Node, Dest, Payload)`. The forwarding *NDlog* rule expresses its traversal through a series of routers until the destination is reached:

```
packet(@NextHop,Dest,Payload) :-
        nextHop(@Node, Dest, NextHop, Cost),
        packet(@Node, Dest, Data), Node!=Dest.
```

One can store annotations either in the packet (i.e. piggyback each tuple with its complete "path" or "provenance"), or maintain state at each router, to allow for subsequent traceback via a distributed query during forensic analysis.

To reduce the storage and communication overhead, there are other proposals such as *ForNet* [19] and *Time Machine*[10] that trade off accuracy for performance, by using summarization (via bloom filters) and sampling techniques respectively to compress the provenance. In addition, to reduce the overhead of queries over the provenance, random moonwalks [21] are used to avoid traversing all possible paths.

3.3 Accountability

Forensics analysis is essentially a form of *call-detail* used in voice telephone networks, where historical information on the caller, callee, length of call, and call status both in real-time and in many cases historically through the examination of call detail records. One important use of the call-detail information is to enforce *accountability*, or proper usage in networks. For example, PlanetFlow [8] is a network auditing service provided on PlanetLab [15], to provide accountability for all traffic generated by PlanetLab services, to ensure that all users are in accordance with PlanetLab policies. With detailed accountability of traffic, more diverse billing is possible as in the telephone network. In addition, increased accountability can also lead to greater incentives for network innovation [11].

3.4 Trust Management

In our final use case, provenance in networks is useful for enforcing trust management policies. For example, a router can drop packets that have traversed via certain paths in order to enforce transit traffic policies. In fact, the path-vector protocol used in BGP carries the entire path during route advertisement, in order to allow ASes to enforce their respective policies.

More generally, beyond routing, provenance enables a recipient node to trace the origins of networked data, and hence enforce trust policies to accept or reject incoming updates based on the source origins. The Orchestra [9] p2p data-integration engine uses provenance in this manner, to accept or reject updates from neighboring nodes by examining the provenance of updates and the trust relationships among nodes. Taking this idea one step further, one can maintain a quantifiable notion of trust, e.g. accepting an update only if over K entities assert the update.

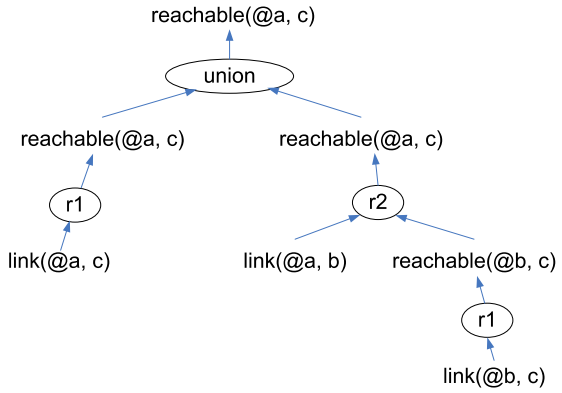

Figure 1: *NDlog* derivation tree for reachable(a,c).

This information can also be encoded in the provenance of an update by maintaining a count of the number of its derivations.

4 Taxonomy of Data Provenance

In this section, we present a taxonomy of data provenance, most of which are derived from existing database literature. We then map that to the use cases presented in Section 3. To illustrate, we make use of an example network which consists of three nodes `a`, `b`, `c` and three unidirectional links `link(a,b)`, `link(a,c)`, `link(b,c)`.

We gives the derivation tree for `reachable(@a,c)` in Figure 1 as a result of executing the *NDlog* query in Section 2.1. This derivation tree essentially represents the lineage or provenance of the tuple, and one can use this tree to figure out the initial input base tuples (at the leaves of the tree). The ovals in the diagram represent the rules (`r1`, `r2`, or `union` to combine their results) that are used for the derivation of `reachable(@a,c)`.

Declarative networks are essentially computations over distributed streams, with time-based sliding windows for soft-state derived tuples. In order to incorporate provenance into distributed streams, we make the following changes to traditional provenance. First, we annotate each derivation with its location (denoted by the location specifier "`@`"). Second, since tuples are soft-state with lifetimes, we also add creation timestamps and time-to-live to the nodes in the tree.

4.1 Local vs Distributed Provenance

The derivation tree shown in Figure 1 can be stored either locally or in a distributed fashion. In *local provenance*, the tree is stored at node `a`, which is the final storage location of `reachable(@a,b)`. In order to have a locally complete provenance, each tuple that is derived needs to piggy-back its entire provenance when shipped from one node to another.

On the other hand, one can utilize *distributed provenance*, which only stores pointers to the previous node to reconstruct its provenance on demand. Hence, node `a` only needs to store the fact that it is derived from `link(@a,b)` which is available locally, and

`reachable(@b,c)` which is stored at node `b`. The analogy here is IP traceback (Section 3.2), where one can either store the entire traversed path within each packet (similar to local provenance), or only maintain enough state at each router to traceback the route on demand.

There are evidently tradeoffs between local and distributed provenance. In local provenance, computation is more expensive for each tuple, but provenance querying is cheap. Also, since each node has the provenance available locally, it can also better enforce trust policies (see Section 3.4). On the other hand, distributed provenance requires no extra communication overhead, but incurs expensive cost of querying the provenance.

4.2 Online vs Offline Provenance

Along another axis, we can further classify provenance as either *online* or *offline*. Online provenance is maintained for network state that is currently valid (i.e. not expired), and *offline* provenance is kept even when the derivations have expired. The purpose of online provenance is for runtime reaction to network anomalies. For example, when a node is detected to be suspicious, one can query the online provenance to delete all routing entries associated with the malicious node.

Online provenance has limited usage given that most networked data are maintained as soft-state with TTLs. In this case, offline provenance is maintained even for data that has long expired. While not useful for real-time diagnostics, it can be used for supporting forensics and enforcing accountability (Sections 3.2 and 3.3). Offline provenance can result in high storage overhead. We will revisit this issue in Section 5.

4.3 Authenticated Provenance

Up to this point, we have assumed that all nodes who compute the provenance are trusted. In practice, authentication is required to ensure the validity of provenance computed by other nodes (e.g. to prevent spoofing of messages from malicious attackers).

Figure 2 shows an alternative derivation tree based on the *SeNDlog* query presented in Section 2.2. We note the following differences. First, since all rule bodies are localized within the context of a security principal, we can omit the location specifiers for each tuple. However, we annotate each operator (denoted by the oval) with the location (or context) where the rule is executed. Second, each node in the tree is asserted by a principal using "says". In an untrusted environment, this means that individual nodes in the provenance tree needs to have digital signatures to validate the authenticity of the computed provenance.

4.4 Condensed Provenance

When computing local provenance, the overhead of shipping the entire provenance with each tuple may be expensive. With authenticated provenance, the overhead is increased due to the digital signatures. We note that in several instances, local provenance is desired (e.g. for deciding whether to accept a tuple based on its origins).

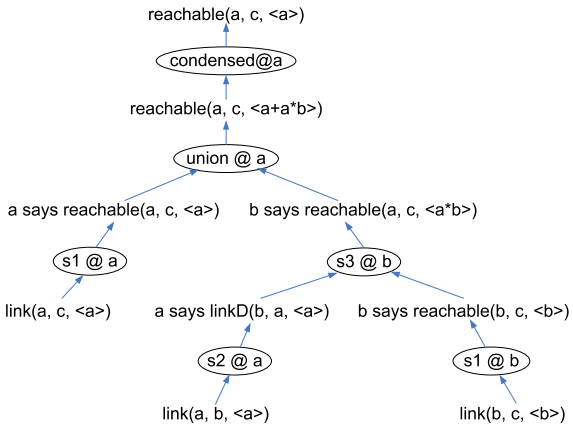

Figure 2: *SeNDlog* derivation tree for reachable(a,c) with annotations for condensed provenance.

To reduce the overhead of computing and sending provenance, we present an existing technique to *condense* the size of local provenance, yet retain sufficient information for enforcing trust based on source origins. This technique is inspired by *provenance semirings* [7] in Orchestra [9] system, where tuples are annotated with provenance expressions that are based on the unique keys of base input tuples. These provenance expressions can themselves be encoded in boolean expressions stored in *Binary Decision Diagrams* (BDD) [5] , and further compressed as presented in [2].

To provide the intuition behind the condensation process, we revisit the derivation tree in Figure 2. Each tuple has an additional field denoted by `<...>` that stores the condensed provenance, where `+` represents union, and `*` represents a join expression. An expression such as `<a+a*b>` for `reachable(a,c)` can be compressed simply into `<a>`. Intuitively, whether the principal `b` is trusted or not is inconsequential given `a`. As long as principal `a` is trusted by the node that receives the `reachable(@a,b)` tuple, this tuple will be accepted, regardless of whether `b` is trusted or not.

4.5 Summary

Revisiting the use cases from Section 3, we summarize the types of provenance that are applicable to each usage scenario. In *real-time diagnostics*, online provenance of existing data is required. The provenance can be local or distributed, and can further be authenticated. On the other hand, *Forensics* and *Accountability* requires offline provenance, and in practice, would be used in conjunction with online provenance. *Trust Management* is best enforced locally at each node, and one can further utilize condensation to reduce the communication overhead.

5 Optimizations

A key challenge in maintaining network provenance is in lowering the storage, communication, and distributed querying overheads. In the previous section, we have seen

how condensed provenance encoded via BDDs can result in a compact representation of provenance that can be evaluated locally for trust management. In addition, we outline three possible optimizations that we would like to further explore as future work:

Proactive vs reactive provenance: In *proactive provenance*, all the provenance of new tuples are eagerly maintained and propagated in the network. In a more *reactive* mode of operation, one can maintain *lazy provenance*, whose computation is triggered only by specified network events. For example, in the earlier path computation example, start computing the provenance of `nextHop` only when slow route convergence is detected. Similarly, offline provenance for forensics can be aged out over time to reduce storage, unless explicitly marked to persist as a result of network anomaly.

Sampling: A straightforward optimization is to only record a portion of the provenance (both online and offline) via sampling techniques. For example, IP Traceback (which generates a new message 1/20,000th of the time) and ForNet (which uses Bloom filters) are examples of this approach. The sampling techniques can also be applied when querying distributed provenance. One example existing technique is the use of random moonwalks [21] to avoid querying all provenance.

Granularity of provenance: In reconstructing network provenance, there are different granularities at which systems can operate. To reduce overhead, provenance can be aggregated and maintained at the AS granularity, which is not conducive for IP traceback. However, we believe that for most network forensic analysis over the Internet, AS granularity is likely to be sufficient for detecting events such as spoofed packet injection.

6 Preliminary Evaluation

In this section, we present a preliminary evaluation study on the overhead of authenticated communication and computing network provenance. We modified the P2 declarative networking system [13] to support the *SeNDlog* query language, which is compiled into distributed dataflows that exchanges messages that are signed with RSA signatures. We further modify various relational operators (particularly joins) in the P2 system to support provenance. In particular, we focus on evaluating the performance of *authenticated provenance* (Section 4.3) which is individually signed by the principal that asserted each fact, and we further apply the condensation (Section 4.4) to reduce communication and storage overhead.

We utilize the OpenSSL v0.9.8b, and Buddy BDD v2.4 libraries to support encryption and provenance. Our experiments are performed on a quad-core machine with Intel Xeon 2.33GHz CPUs and 4GB RAM running Fedora Core 6 with kernel version 2.6.20. In our experiments, we execute up to 100 P2 processes representing different nodes on the machine.

For the query workload, we utilize the *Best-Path* recursive query that computes the shortest paths between all pairs of nodes. This query is obtained from the *NDlog* all-pairs reachability query presented in Section 2, with additional predicates to compute the actual path, cost of the path, and two extra rules for computing the best paths. As input, we insert link tables for up N nodes with average outdegree of three, and vary the size of N from 10 to 100.

To isolate the individual overhead of authenticated communication and provenance, we execute three versions of the *Best-Path* query: *NDlog* version without authentication and provenance, *SeNDLog* with authentication but without provenance, and *SeNDLogProv* with both authentication and provenance. Our metrics of evaluation are as follows:

- **Query completion time (s):** Time taken for a query to finish execution. As our example programs are recursive, this means the time elapsed before the system reaches a distributed fixpoint, where all nodes finish computing their best paths.
- **Bandwidth usage (MB):** The total combined bandwidth usage across all nodes required for executing the distributed query.

In our experiments, we measure the computation and bandwidth overheads of encryption and provenance by comparing *NDLog*, *SeNDLog* and *SeNDLogProv*. Figure 3 and 4 shows the query completion time and bandwidth utilization respectively, averaged over 10 experimental runs. We summarize our results as follows:

***SeNDlog* overhead:** The use of authenticated communication in *SeNDLog* incurs in the average 53% delay in query completion time and 36% bandwidth utilization compared to *NDlog*. As N increases, the additional overhead decreases. For example, when N is 100, the overhead is 44% and 17% respectively. Given that we are running multiple P2 processes on a single node and generating a signature for each tuple, this represents an upper bound on the encryption overhead.

Condensed provenance overhead: The query completion time of *SeNDLogProv* increases by 41% compared to *SeNDLog* due to the overhead of computing and shipping provenance. In addition, *SeNDLogProv* requires 54% more bandwidth than *SeNDLog*. Similar to the *SeNDlog* overhead above, we observe that provenance overhead decreases as the number of nodes increases. For example, when N is 100, *SeNDLogProv* only incurs additional 6% and 10% costs in computation and bandwidth overhead respectively. Our results demonstrate that the BDD-encoded condensed provenance is efficient for recording derivation of tuples, at reasonably low overhead especially for larger networks.

7 Conclusion

In this paper, we argue that network accountability and forensic analysis can be posed as data provenance computations and queries over distributed streams. In particular, one can utilize provenance-aware declarative networks with appropriate security extensions to provide a flexible

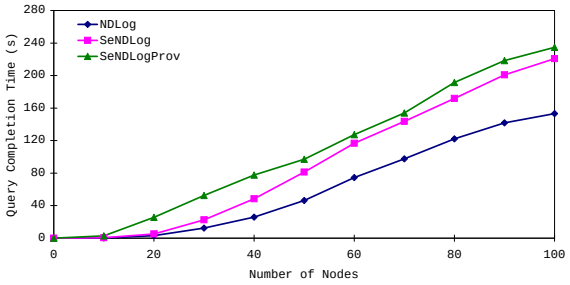

Figure 3: Query completion time (s) for *Best-Path* query

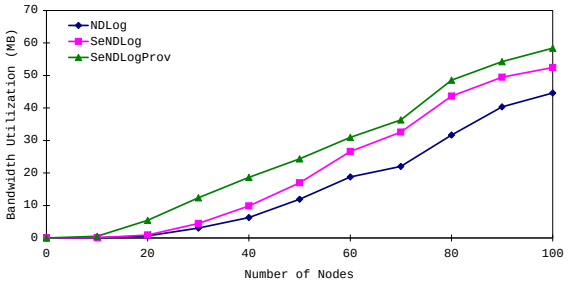

Figure 4: Bandwidth utilization (MB) for *Best-Path* query.

declarative framework for specifying, analyzing and auditing networks. To prove our case, we propose a taxonomy of data provenance along multiple axes, and show that they map naturally to several use cases ranging from network forensics and diagnostics to trust management. We suggest techniques to efficiently compute and store network provenance, and provide an initial performance evaluation using the P2 declarative networking system.

Our future work is proceeding along several fronts. First, while we focus on forensics and accountability over the Internet, we intend to explore the general applicability of these techniques to overlay networks and sensor networks. Second, we are in the process of evaluating a variety of secure networks specified and implemented using *SeNDlog* (e.g. secure Chord routing, DNSSEC), and studying the usage of network provenance for a variety of networks. This will enable us to investigate cross-layer analysis opportunities that arise as a result of having a single integrated system that unifies network and security specifications.

References

[1] M. Abadi and B. T. Loo. Towards a Language and System for Secure Networking. In *NetDB*, 2007.

[2] Anonymous. Paper is under submission.

[3] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing packet obituaries. In *Proc. of 2006 ACM SIGCOMM Workshop on Mining Network Data (MineNet '06)*. ACM Press, Sept. 2006.

[4] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee. Accountability as a service. In *USENIX Steps to Reducing Unwanted Traffic on the Internet*, 2007.

[5] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3), 1992.

[6] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[7] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *ACM Symposium on Principles of Database Systems*, 2007.

[8] M. Huang, A. Bavier, and L. Peterson. PlanetFlow: Maintaining accountability for network services. *Communications of the ACM*, 32(6):89–94, June 1989.

[9] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, January 2005.

[10] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *Internet Measurement Conference (IMC)*, 2005.

[11] P. Laskowski and J. Chuang. Network monitors and contracting systems: Competition and innovation. In *Proceedings of ACM SIGCOMM Conference on Data Communication*, 2007.

[12] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking: Language, Execution and Optimization. In *ACM SIGMOD*, June 2006.

[13] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing Declarative Overlays. In *ACM SOSP*, 2005.

[14] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. In *ACM SIGMOD*, 2005.

[15] PlanetLab. Global testbed. 2006. http://www.planet-lab.org/.

[16] R. Ramakrishnan and J. D. Ullman. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming*, 23(2):125–149, 1993.

[17] S. Raman and S. McCanne. A model, analysis, and protocol framework for soft state-based communication. In *SIGCOMM*, pages 15–25, 1999.

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of ACM SIGCOMM Conference on Data Communication*, 2000.

[19] K. Shanmugasundaram, N. Memon, A. Savant, and H. Bronnimann. ForNet: A distributed forensics network. In *Proc. of 2nd International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, 2003.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM Conference on Data Communication*, 2001.

[21] Y. Xie, V. Sekar, M. K. Reiter, and H. Zhang. Forensic analysis for epidemic attacks in federated networks. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, pages 43–53. IEEE Computer Society, May 2001.