NetTrails: A Declarative Platform for Maintaining and Querying Provenance in Distributed Systems

Wenchao Zhou* Qiong Fei* Shengzhi Sun* Tao Tao*
Andreas Haeberlen* Zachary Ives* Boon Thau Loo* Micah Sherr◇
*University of Pennsylvania, Philadelphia, PA ◇Georgetown University, Washington, DC
{wenchaoz, qiongfei, shengzhi, taot, ahae, zives, boonloo}@cis.upenn.edu,
msherr@cs.georgetown.edu

ABSTRACT

We demonstrate *NetTrails*, a declarative platform for maintaining and interactively querying *network provenance* in a distributed system. Network provenance describes the history and derivations of network state that result from the execution of a distributed protocol. It has broad applicability in the management, diagnosis, and security analysis of networks. Our demonstration shows the use of NetTrails for maintaining and querying network provenance in a variety of distributed settings, ranging from declarative networks to unmodified legacy distributed systems. We conclude our demonstration with a discussion of our ongoing research on enhancing the query language and security guarantees.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed Applications*; E.1 [**Data Structures**]: Distributed Data Structures; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical User Interfaces*

General Terms

Design, Management

Keywords

Provenance, Visualization, Declarative networking

1. INTRODUCTION

Distributed systems are increasingly being deployed at Internet scale for a variety of applications. With functionalities as diverse as content-distribution networks, data analytics, p2p search, and network monitoring, these complex and often multifaceted distributed systems impose significant challenges, particularly in the areas of diagnostics and forensics. We show how a variety of network tasks (for example, identifying misbehaving users and enforcing trust policies) can be achieved by maintaining and analyzing *network provenance* [10], which captures the history and derivations of network state that result from the execution of a distributed protocol. We demonstrate *NetTrails*, a declarative platform for incrementally maintaining, interactively navigating, and querying network provenance in a distributed system.

Figure 1 shows an overview of the NetTrails system on a single node. On the left side of the figure is the application whose network provenance is being tracked by NetTrails. The application can either be a declarative network or an unmodified legacy application ("black box") that utilizes a *proxy* to extract state changes from intercepted application messages.

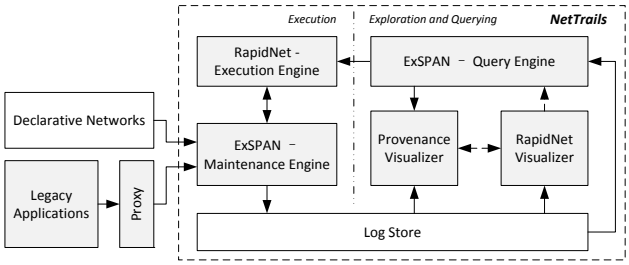

Figure 1: Architectural overview of NetTrails.

NetTrails integrates features from the RapidNet [4, 7] declarative networking engine, the ExSPAN [10] network provenance engine, and an interactive provenance and network topology visualizer for exploring and issuing customized provenance queries on network state. The goals of this demonstration are: (1) to demonstrate the query processing and optimization techniques used in a declarative networking engine to support network provenance in a distributed environment, (2) to show concrete use cases of the system based on traditional declarative networking protocols [3] as well as an unmodified legacy application, the popular Quagga network routing suite [6], and finally, (3) to highlight our ongoing research work on extending the system to enable new network analysis capabilities.

2. SYSTEM OVERVIEW

We provide an overview of NetTrails by describing the system's major components and the mechanisms by which they interact.

2.1 RapidNet Declarative Networking Engine

NetTrails utilizes a declarative networking engine to implement declarative protocols, and it provides distributed querying capabilities to maintain and query provenance information in a distributed fashion. Declarative networks are specified using *Network Datalog* (*NDlog*), a distributed recursive query language used for querying network graphs. *NDlog* queries are executed using a distributed query processor and are continuously maintained as distributed views over existing network state. Previous work [3] has demonstrated that a variety of distributed systems, such as path-vector routing, distance-vector routing, and the Chord distributed hash table, can be specified and implemented in NDlog in orders of magnitude less lines of code than imperative implementations.

The specific engine that we have adopted is RapidNet [4], a declarative networking engine that is integrated with the new

SIGMOD'11, June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

ns-3 [5] network simulator. *NDlog* programs are compiled into ns-3 code (in C++) using the RapidNet compiler and executed natively. The compiled code can either run as an ns-3 application in simulation mode, or as an actual implementation that can run directly on a network. The system has been used to deploy distributed systems on PlanetLab and on the ORBIT wireless testbed. RapidNet further includes a network visualization tool. RapidNet is open source software and is available for download [7].

2.2 ExSPAN Network Provenance Engine

ExSPAN [10] consists of two components. The *maintenance engine* takes as input either *NDlog* programs or input/output dependencies captured from legacy applications, and then incrementally computes and maintains network provenance information as distributed relational tables. Second, the *distributed query engine* executes user-customizable provenance queries that are evaluated across multiple nodes. During the protocol execution, ExSPAN [10] (Section 2.2) incrementally maintains provenance information using RapidNet [4] as its distributed query engine. Our architecture offers a unified framework, as both maintenance and querying functionalities are specified as *NDlog* programs.

Provenance model: In ExSPAN [10], the provenance graph is internally maintained as relational tables, which are distributed and partitioned across all the nodes in the network. Network provenance is modeled as an acyclic graph $G(V,E)$. The vertex set V consists of *tuple vertices* and *rule execution vertices*. Each tuple vertex in the graph is either a base tuple or a computation result, and each rule execution vertex represents an instance of a rule execution based on a set of input tuples. The edge set E represents dataflows between tuples vertices and rule execution vertices.

Provenance maintenance: Since all data dependencies are explicitly captured in derivation rules written in *NDlog*, the provenance information that belongs to a declarative network can be captured easily. In previous work [10], we have presented an automatic rule rewriting algorithm that takes as input a *NDlog* program and outputs a modified program that contains additional rules for capturing the program's provenance information. These additional rules define network provenance in terms of views over base and derived tuples. As the network protocol executes and updates network state, views are incrementally recomputed.

In the case of a legacy application, capturing provenance information requires some additional work – particularly if the source code of the legacy application is unavailable. In such cases, we utilize *NDlog*'s concept of *"maybe" rules*, which describe possible causal relationships between messages entering and leaving the legacy application. In contrast to ordinary derivation rules, the output tuple of a "maybe" rule is not necessarily always derived (depending on internal decisions in the legacy application). To illustrate this, we consider a legacy router used in Internet routing. The following "maybe" rule captures the likely causal relationship between incoming and outgoing route advertisements:

```
br1 outputRoute(AS,R2,Prefix,Route2) ?-
     inputRoute(AS,R1,Prefix,Route1),
     f_isExtend(Route2,Route1,AS)=1.
```

Rule `br1` is an example of a "maybe" rule (syntactically recognized via the `?-` symbol), capturing possible dependencies between an `inputRoute` tuple that arrives at a router and an `outputRoute` that is subsequently generated by the router. This matching can be achieved by exploiting knowledge about iterdomain routing: a router that applies the standard interdomain routing protocol prefixes its address to incoming route advertisements before exporting the selected routes to its neighbors. The `f_isExtend(R1,R2,N)` function leverages this by checking whether routes `R1` and `R2` differ only by the addition of the node identifier `N`. If so, the "maybe" rule infers a causal relationship.

Provenance querying: Once generated, network provenance can be queried by issuing distributed queries. Since provenance information is distributed across nodes, query execution performs a traversal of the provenance graph in a distributed fashion. ExSPAN allows users to customize the provenance queries. For instance, users can query for a tuple's lineage, the set of all nodes that have been involved in the derivation of a given the tuples, and/or the total number of alternative derivations. To reduce querying overhead, ExSPAN adopts a set of optimization techniques [10], which include caching previously queried results, leveraging alternative tree traversal orders, and performing threshold-based pruning.

2.3 Interactive Visualization

Although NetTrails is designed to execute in a distributed environment, some state needs be centralized to facilitate the visualization of provenance queries and results. In particular, per-node provenance information and other system state (such as the network topology and bandwidth utilization) can be periodically captured as system snapshots at each node, and then propagated to a central *Log Store* that resides at the visualization node. These logs are subsequently used for interactive visualization, query, and replay during our demonstration.

The generated logs are replayed using the *RapidNet visualizer* (to show the actual network topology, and position of nodes and links as the topology changes) and a *provenance visualizer*, which is based on hypertrees [1]. The provenance visualizer provides two useful features: the provenance graph is presented on a hyperbolic plane, enabling users to focus on small segments of the graph; additionally, users can navigate the provenance graph by changing focus with smooth transitions by clicking on or dragging the screen.

Figure 2 shows a series of screenshots of the visualizer while it is used to interactively navigate the tree structure. This example is based on the MINCOST protocol, which computes pair-wise minimal path costs in a network. Figure 2(a) shows the root of the provenance tree at a particular point in time; Figure 2(b) shows all the pair-wise minimal path costs; and Figure 2(c) shows a close-up view of a particular tuple, as well as its attribute values and location (shown in the black rectangle).

Note that the topology and the provenance visualizer are interactively navigated in tandem: during the replay, users can interactively pause the network at a given time, and then view the provenance information of any node. Similarly, by navigating the hypertree provenance to explore dependencies among nodes, users can traverse and view the network state and the rules executing at another node. At any point in time, the users can customize the tree by issuing a provenance query that is then evaluated by ExSPAN, potentially across several nodes.

3. DEMONSTRATION PLAN

NetTrails visualizes the topologies and statistics of the distributed system (in the RapidNet visualizer) and its corresponding provenance information (in the provenance visualizer) based on the execution traces and provenance snapshots taken during the system execution. To illustrate this, Figure 3 shows an example execution of the current version of our demonstration where we show the provenance of the system state (captured as tuples) for a running MINCOST program. One may further issue customized queries against the provenance and visually show their progressive steps.

We plan to further extend and continue the development of NetTrails, and demonstrate the following two use cases:

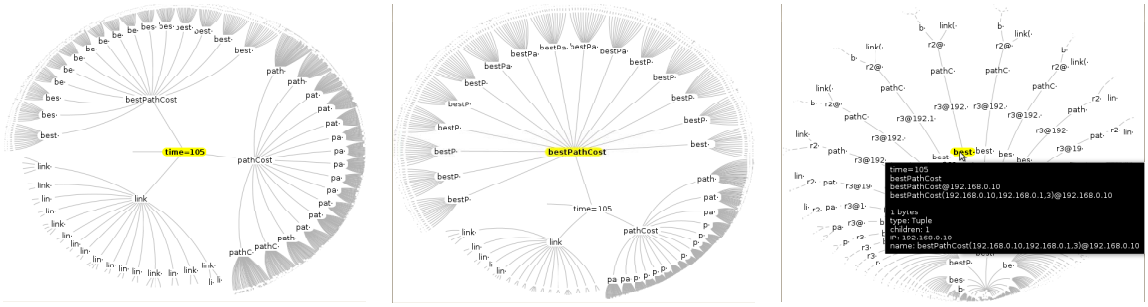

Figure 2: An example of interactive exploration in the provenance visualizer. Users start from the system-wide snapshot of the provenance at time T (screenshot a), select the table that they are interested in (screenshot b), and finally locate the provenance of a particular tuple instance (screenshot c). Focus changes are connected by smooth transitions, enabling progressive exploration.

Declarative networks: Our first use case focuses on distributed systems implemented using the declarative networking framework, including the MINCOST protocol, the path-vector protocol, and dynamic source routing (DSR). This use case demonstrates the applicability of NetTrails in a variety of declarative networks running in different environments (e.g. static vs mobile network). In each configuration, we show that NetTrails correctly captures and maintains provenance, as network state is incrementally recomputed as the underlying network topology changes.

Legacy applications: Our second use case explores the integration of NetTrails with legacy applications. We use the Quagga routing suite [6] to set up a number of BGP (*Border Gateway Protocol*) instances in multiple ASes. BGP is the standard interdomain routing protocol used by all Internet ISPs to exchange routing information with one another. To emulate an actual network environment, we instantiate all Quagga BGP daemons on a single machine and use the proxy to intercept BGP messages. The Quagga instances form a topology of ASes that consists of several large and small ISPs connected by a mix of customer/provider/peer relationships. Using actual BGP traces from RouteViews [8], we show that NetTrails can capture derivation histories and origins of routing entries.

We demonstrate that, with the provenance information captured in NetTrails, users can perform various analytical and diagnostic tasks simply by navigating in the provenance visualizer. Examples include tracing back from root causes, monitoring cascading effects that result from network topology updates, and determining the parties that have participated in the derivation of a tuple. One may even monitor how system state updates lead to changes in provenance, to understand the effects of these updates.

Users can also access the provenance information by issuing custom queries directly to ExSPAN. We demonstrate different types of provenance queries, such as querying the set of contributing base tuples, participating nodes, or the total number of derivations. We also visually demonstrate that optimization techniques, such as caching and threshold-based pruning, effectively reduce the network traffic.

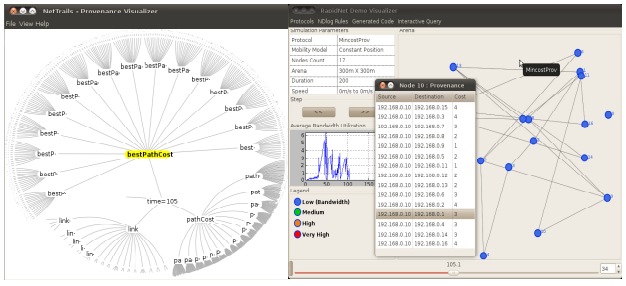

Figure 3: A screenshot of the NetTrails demonstration.

Finally, we use our demonstration as a basis for discussing some of our ongoing work: (1) exploring distributed variants of graph-based provenance query languages such as ProQL [2] for formulating queries and transformations over network provenance data, and (2) enhancing the current system to securely utilize network provenance information in untrusted environments and enable efficient explanation of the causes and effects of network state [9].

4. ACKNOWLEDGMENTS

This work was supported by NSF grants IIS-0477972, IIS-0713267, CNS-0721541, IIS-0812270, CCF-0820208, CNS-0845552, CNS-1040672, CNS-1054229, AFOSR MURI grant FA9550-08-1-0352, DARPA award N66001-11-C-4020, and NPS award N00244-11-1-0008.

5. REFERENCES

[1] Hyperbolic Tree Java Library. http://sourceforge.net/projects/hypertree/.

[2] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD*, 2010.

[3] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. In *CACM*, 2009.

[4] S. C. Muthukumar, X. Li, C. Liu, J. B. Kopena, M. Oprea, and B. T. Loo. Declarative toolkit for rapid network protocol simulation and experimentation. In *SIGCOMM-demo*, 2009.

[5] Network Simulator 3. http://www.nsnam.org/.

[6] Quagga Routing Suite. http://www.quagga.net/.

[7] RapidNet. http://netdb.cis.upenn.edu/rapidnet/.

[8] RouteViews project. http://www.routeviews.org/.

[9] W. Zhou, A. Haeberlen, B. T. Loo, and M. Sherr. Tracking Adversarial Behavior in Distributed Systems with Secure Network Provenance. Technical Report MS-CIS-10-28, University of Pennsylvania, 2010.

[10] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *SIGMOD*, 2010.