# QUERYING EVENTS IN DISTRIBUTED SYSTEM THROUGH PROVENANCE

**ALI ZAFAR SADIQ**

**BSSE 0603**

Academic Project Report

Submitted to the Bachelor of Science in Software Engineering Program Office

of the Institute of Information Technology, University of Dhaka

in Partial Fulfillment of the

Requirements for the Degree

**BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING**

Institute of Information Technology

University of Dhaka

DHAKA, BANGLADESH

© Ali Zafar Sadiq, 2017

QUERYING EVENTS IN DISTRIBUTED SYSTEM THROUGH PROVENANCE

ALI ZAFAR SADIQ

Approved:

*Signature* *Date*

Supervisor: Dr. Kazi Muheymin-Us-Sakib

*Professor*

# Abstract

Debugging in Distributed Systems possess great challenges as the state of the system continuously changes. For this reason, different approaches are adopted to identify the root cause by analyzing the system and to make it interactive and helpful for the network operators. Provenance is one of the techniques where information related to systems are stored and whenever any problem arises that information used to track the root problem in the system. Currently, there are many research works and tools to display events in Distributed Systems that happened by backtracing using provenance information. However recently Negative Provenance added a new dimension to this research area and is yet to be explored more, for explaining complex events that did not happen.

In the existing *Y!*, the concept of negative provenance was introduced in declarative networking. It explored some cases and explained the importance of exploring the concept of negative provenance in debugging Distributed Systems. In this report a tool is proposed to query negative provenance. NQ tool can answer both positive and negative events in virtually created Distributed System environment using rapidnet [1]. Different test environment was created using NDlog rule, and was executed with ns3 and rapidnet. Based on the log file produced from rapidnet, the events can be queried using rapidnet about why they happened or why they didn't. And the provenance graph of the event is visualized for getting detail about the reason behind occurrence of events.

The main achievements of this work is visualization of provenance graph, improvement of time of response and successful implementation of the existing *Y!* [2].The time required by existing approach for diagnosing off path change in BGP was about .15 second however in the proposed tool it is about.04 seconds, and in diagnosing link failure, porposed tool took about .02 seconds whereas existing tool took about .1

second. Improvement of time happened because of keeping the formatted logs in memory rather than in disk. And visualization is added for better understanding of the query responses.

# Acknowledgement

Firstly, I would like to express sincere gratitude and appreciation to Dr. Kazi Muheymin-Us-Sakib and Asif Imran for their guidance and support. Their inspiration helped me to work hard and produce a quality work. I would also like to thank committee members for their guidance during the presentations. I would also like to thank my classmates as well as all staffs of IIT for their help and support.

# Contents

Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii

Acknowledgement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

**1 Introduction** **1**

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.3 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.4 Organization of the Report . . . . . . . . . . . . . . . . . . . . . . 5

**2 Background Study** **7**

2.1 Provenance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.2 Provenance overhead . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Networking and Declarative Networking . . . . . . . . . . . . . . . 9

2.4 Distributed Systems . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**3 Literature Review** **13**

3.1 Declarative Networking . . . . . 14

3.2 Distributed System Debugging . . . . . 15

3.3 Data Provenance . . . . . 17

3.4 Data oriented workflow Provenance . . . . . 19

3.5 Provenance In Distributed Systems . . . . . 20

3.6 Negative Provenance . . . . . 22

3.7 Summary . . . . . 23

**4 NQ : A tool for Querying in Distributed System Positive and Negative Events through Provenance** **24**

4.1 Overview of the proposed Querying Technique . . . . . 25

4.2 Description of System Architecture . . . . . 25

Protocol and invariant specification . . . . . 25

Logs and formatting . . . . . 27

Query Processor . . . . . 29

Provenance Graph . . . . . 31

Visualization . . . . . 36

4.3 Summary . . . . . 37

**5 Implementation and Result Analysis** **38**

5.1 Implementation Details . . . . . 39

5.2 Experimental Datasets . . . . . 40

5.3 Result Analysis . . . . . 43

5.4 Comparative Analysis . . . . . 45

5.5 Summary . . . . . 46

**6 Conclusion** **47**

6.1 Discussion . . . . . 47

6.2 Threats to validity . . . . . 49

6.3 Future Work . . . . . 49

References . . . . . 51

# List of Figures


2.1 Client Server Architecture . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.2 N-tier Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.3 Peer to Peer Architecture . . . . . . . . . . . . . . . . . . . . . . . . 11

4.1 Architecture of the NQ . . . . . . . . . . . . . . . . . . . . . . . . . 26

4.2 NS3 stimulation with specified protocols . . . . . . . . . . . . . . . . . 27

4.3 BGP datalog used in case study . . . . . . . . . . . . . . . . . . . . . 28

4.4 Getting Logs used in case study . . . . . . . . . . . . . . . . . . . . . 29

4.5 Processing queries . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

4.6 Visual Display of provenance graph . . . . . . . . . . . . . . . . . . . 36

5.1 A BGP Topology used in off path change. . . . . . . . . . . . . . . . . 41

5.2 A BGP Topology used in BGP blackhole query. . . . . . . . . . . . . . 41

5.3 Disk Consumption . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

# Chapter 1

# Introduction

Today Distributed Systems are at the heart of Cloud Computing. More specifically, cloud computing is one of the most recent innovations of Distributed Systems which shifts the computing control and hassles for the general users, providing them a field of abundant computing resources. However, the cloud is susceptible to a number of attacks and resource failures. Analyzing provenance can be an important mechanism to detect attacks in the cloud and at the same time monitor resource malfunctions. Provenance helps to explain the difference (that is series of actions which led to the change) of an object at its current state from that of its origin. So tracking and then analyzing preceding events helps in diagnosing actual reason behind system failure or security breaches. Again, the absence of events like loss of packet data or unavailable routes can be explained by using counterfactual reasoning to identify conditions under which these events could have occurred [2]. However, there may be situations where a tampered node may give wrong information. In such adversarial cases, provenance can be used to track manipulation or tamper-evident properties from neighboring nodes to assist operators to detect compromised nodes [3]. So provenance information of events provides useful data to analyze, optimize and secure any system.The graphical representation of provenance or provenance graph is built from provenance

information for better and easier understanding about different lineage and derivation information. In this chapter, there are sections on motivation, research questions, contribution of the work and at the end contains the organization of this report.

## 1.1 Motivation

Due to constant changes, Distributed Systems are complex. In these complex systems, finding bugs and errors are difficult as various variables and conditions of the system are constantly changing. A large number of works and tools are being developed using provenance, to help operators debug the Distributed Systems [2] [4] [5]. However, all these gave information about what happened but not why did not happen. Recently *Y!* [2] added a new dimension in this field, considering conditions for which positive events happen, events that did not happen could also be explained. The main motivation of this work is to explore negative provenance and contribute further to debugging using provenance.

Provenance is metadata that can explain the root cause of any event. Explaining reason behind these missing events requires querying preceding events to find the root cause which would produce the required missing event. The explanation of any event is limited by available information and can only explain why it occurred or did not occur. Again due to not having any notion of program semantics there is a problem in finding the main problem caused by buggy programs (like software defined networking). In order to find a missing or negative event in a node, it needs to consider all possible positive events from the node and its neighboring nodes that would have resulted in the particular missing event [2] and from them the reason can be explained. So a large chain of events needs to be considered for explaining negative provenance.

After collecting provenance information, provenance graph can be constructed [3]. It represented as a directed acyclic graph in which vertices are events and edges indicate causal relationships between them. As provenance graph from the command line can give may become complex for any network operator, so visualizing provenance graph gives a good overview of the situation happening in a complex distributed system. Different works are going on to make the provenance graph as user friendly as possible.

In cases where Distributed Systems are in a suspicious state and if there are the faulty or misbehaving nodes, provenance information can play a vital role to debug and identify them [3]. It can also be used to assess the damages that they (that is faulty nodes) might have caused to the system. Again, provenance can be used to answer audit questions, to predict future workloads. Based on such predictions, any system's performance, as well as the availability of service, can be identified. Researchers have also conducted work to reason about program changes and leading to a potential list of solutions which would help an operator to find the fix system faults quickly [6]. Also to improve how provenance can be used to query Distributed System using numerous optimization techniques and thus reducing bandwidth costs were also investigated in a research work [4].

As the state of any system through provenance information can be better understood and used to answer about system's behavior corresponding to factors and variables influencing the system, so it has immense importance in Distributed Systems which deals with lots of information. Through provenance detection and effective querying, Distributed Systems will achieve better results in fault diagnosis, performance, and security which are the main concerns of this field. Currently, negative provenance has added a new dimension in this field by trying to explain events that did not happen. So both positive and negative provenance needs to be further explored for better diagnosing and debugging.

## 1.2 Research Questions

In order to explain the provenance of any event, we need store provenance information and operate query on them. Existing tools like *Y!* [2] can explain the negative or missing events, however, are limited to the information available(as explanation requires metadata), has no notion of program semantics and was not done in a large setting. So it leads to following research questions

> **How to effectively identify occurrence and missing of events in Distributed System and show the result visually through provenance graph?**

In order to identify the reasons we need provenance information and identify negative events which require enquiring all positive events from which it might have originated. So, this research question can be answered in form of some sub-questions. It can be answered by the following sub-questions. Sub Problems

1. How to identify negative events from available provenance information

   To answer this sub-question following steps will be adopted

   (a) To use counterfactual reasoning to examine all possible causes that could have produced the missing effect.

   (b) To sort and identify the actual reason, either by reaching a positive event that can be traced with normal provenance, or a negative root cause (such as missing entries in a configuration file).

2. How to show query result of an event in a interactive and user friendly way?

   To answer this sub-question following steps will be adopted

   (a) Processing the query to find the actual cause of an event.

(b) Finding out the responsible events and representing them in a compact way.

(c) Displaying the processed query in a graphical way (that is using provenance graph).

## 1.3 Contribution

The main contribution that in this work is improvement of response time, implementation of the existing approach and then modifying it and lastly the graphical display of the provenance graph. This work can explain both positive and negative events. As positive provenance explains what happened, negative provenance uses the positive provenance to explain events that did not happen. In the Distributed Systems, debugging process is complex and interpretations are similarly tough for network operators. Visualizing the process indeed helps to better diagnose the faults in the complex Distributed System.

The result analysis from comparative study shows that the response time is improved however tbe number of response vertices that explains the reason behind present or absence of any event is more than existing. The increased response vertices may cause the network operator long time go through the vertices, but it also explains the facts in depths.So overall the performance of the work seems to be acceptable.

## 1.4 Organization of the Report

The overview about remaining chapters of this report is mentioned here. Chapters are organized in the report as following.

**Chapter 2:** Provenance, provenance overhead, Distributed Systems, networking and declarative networking are briefly described here. This chapter gives the basic foundation concepts required to understand the work.

**Chapter 3:** Existing literature works on Distributed System debugging, declarative networking, data provenance, provenance in Distributed Systems, negative provenance are discussed in this chapter.

**Chapter 4:** The proposed approach of NQ is explained in this chapter. Algorithms and procedure, that are being used, are mentioned in this chapter.

**Chapter 5:** The implementation and comparison of NQ and existing tool *Y!* is mentioned here.

**Chapter 6:** In this last chapter discussion about proposed approach and future plans are mentioned.

# Chapter 2

# Background Study

In this age of information, use of Distributed Systems is getting popular due to their ability process huge amount of data efficiently and inexpensively. Distributed Systems are complex by nature because of the complex network of routers, hosts, and middleboxes. Finding the actual reason for a problem in such system is challenging. The problem can even manifest in subtle ways that have no direct relation to the root cause. So it is convenient to have network debugger to assist network operators with the problem.

In this chapter, the architecture of distributed computing, system debugging, assumptions and pre-requisites of the research have been analyzed. Provenance is used for debugging system and getting reasons behind errors or unwanted behavior. Since Distributed Systems are complex, provenance can help to analyze system problems and give appropriate solutions.

## 2.1 Provenance

Provenance is the metadata that contains the history of data objects, processes or operations from the beginning to its current state. In a Distributed System, all information regarding the creation time, the modifier and the process used to modify the stored data or the processes running are collected as provenance. Using this information, system administrators and data forensic experts validate data and ensure reliability [7].

Recognizing the importance of provenance, a large number of researches are being conducted on detection, storage, representation and analysis of correctness of provenance data from information captured real time or stored in provenance repositories. Many research organizations use provenance information collected from distributed applications to analyze and obtain answers to research questions. These research organizations host those provenance data so that other research communities can use those data in their experiments.

Currently, data sharing across heterogeneous environments are becoming more common. so the need to query or reassemble provenance from multiple data sources is becoming increasingly important. So Distributed Systems should provide the ability to collect and store provenance for all operations executed at the physical level as well as the virtual level for any distributed application.

## 2.2 Provenance overhead

Provenance or metadata about system activities and operations are usually stored. The amount of storage and computation overhead required to inquire about system state using provenance depends to what level information is wanted [8]. Especially

in scientific databases where there are large datasets and in high computation areas, provenance overhead might cause an unwanted effect on system architecture.

Currently, provenance is stored and replayed to debug system and identify the root cause of problems. The effective process of storing a unit piece of provenance information is still to be developed. Again provenance should be reliable and represented in a user friendly way. Provenance mechanism that will not affect the system architecture and performance is desired.

## 2.3 Networking and Declarative Networking

A digital telecommunications network where data processing nodes share and exchange resources between each other using data link is known as computer networking [9]. Connections between nodes are established using either cable media or wireless media. The data processing nodes may be computers, nodes, servers, networking hardware etc. Two such devices can be networked together whether they have a direct connection or not. Connection to the network is established using either cable media or wireless media.

Declarative Networking is the programming methodology where developers can concisely specify protocols. It makes use of declarative query language to specify and implement network protocols. The main aim of declarative networking is to simplify specifying, implementing, deploying and evolving network design. As for example of such systems are P2 system [10] and Rapidnet [1]. Using Declarative Networking, various network protocols and architectures can be set up and various experiments can be conducted. Rapidnet allows setting up such environments virtually using the ns3 [11] network stimulator and rapidnet libraries. Using declarative networking will surely usher new paradigm in networking as protocols can be concisely specified and

changed whenever needed. So different network protocols and topologies can be tested virtually using rapidnet and ns3.

## 2.4 Distributed Systems

Distributed Systems are models in distributed computing where networked computers communicate and coordinate their actions by passing messages. A Distributed System may have a common goal, such as solving a large computational problem by considering a collection of autonomous processors as a unit or by coordinating the use of shared resources. Components in Distributed Systems communicate with each other to achieve the common goal.

In distributed computing various hardware and software architectures are used. At a lower level, multiple CPUs are interconnected with some sort of network. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system. Distributed programming typically falls into one of the several basic architectures: client server, three-tier, n-tier, or peer-to-peer.

- Client Server: In this network architecture, each computer or processes are either client or server. Provider of service or resources are servers and service requesters are clients. Few examples of this architectures are email, network printing, and World Wide Web [12]. Figure 2.1 represents a client server architecture.
- N-tier : In N-tier architecture, software is engineered to have processing, data management, and presentation functions physically and logically separate. This is done so that services can be delivered at top capacity and to provide easy management facility [13] . In Figure 2.2 a N-tier architecture is shown.

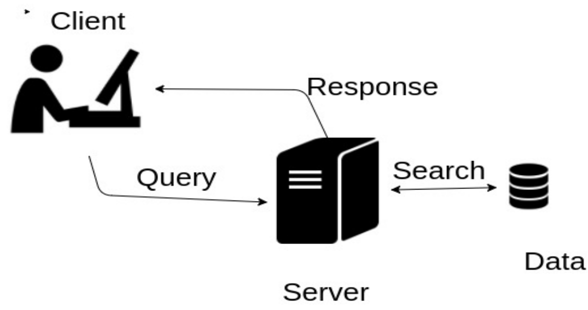


Figure 2.1: Client Server Architecture

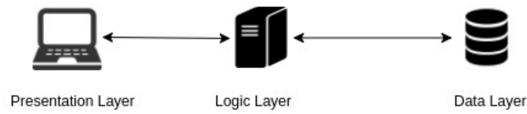


Figure 2.2: N-tier Architecture

- Peer to Peer : In this architecture, all machines communicate with each other and share their workloads. Peers can serve both as clients and as servers and each are independent as those have their own storage and computing power [14]. Figure 2.3 represents a peer to peer architecture.

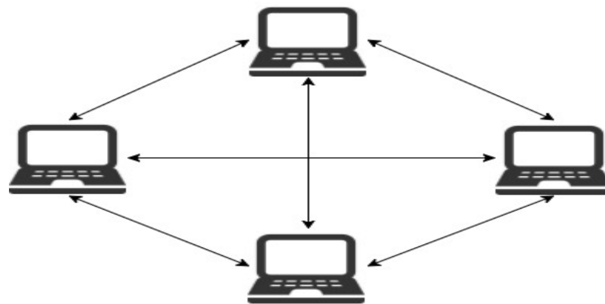

Figure 2.3: Peer to Peer Architecture

## 2.5 Summary

Currently, a large number of computations are done using Distributed Systems. The use efficient use of computational power by distributing the workload was and still is an active area of research. However, there may be data errors and security vulnerabilities due to distributed nature of the Distributed Systems. Provenance tries to track events to get to the root cause and thus identifies the reason behind problems. So in Distributed Systems application of provenance is increasing day by day. In the following chapter, related papers and journals are reviewed to enhance knowledge and get a better understanding of relevant works in the current domain of work.

# Chapter 3

# Literature Review

Existing research work on provenance and negative provenance are focused in this chapter. Research issues in provenance detection and validation, provenance in Distributed System debugging and applications are also discussed. As this is the age of information, data and information are being highly valued. Effective provenance detection ensures integrity and validity of the data as well as how data has been manipulated up to its current state. Using this ability of provenance to explain state change, currently it has been a topic of extensive research in Distributed Systems where system state always changes. Provenance in distributed explains the change of system and the reason behind the changes. Various fault diagnosing and debugging tools as well as automatically fault fixing systems are being researched and developed using provenance. Although, through positive provenance, it can be explained why anything changed or happened, but not why something did not happen. So negative provenance helps in explaining facts more than positive provenance. However, research work on negative provenance are scanty and experiments were done on few test cases. So more experiments are needed in the field of negative provenance in a Distributed System.

## 3.1 Declarative Networking

Due to increasing diversity in network architectures over the past few decades, the design of new protocols is becoming a subject of interest. However network protocol design and implementation is challenging by considering facts of extensibility, flexibility, efficiency and robustness. Currently, in order to change or upgrade any deployed routing protocols, access to each router is needed to modify its software. This process is made even more tedious and error-prone by the use of conventional or imperative programming languages.

Boon Thau Loo et al applied database query-language and processing techniques into networking domain and introduced the concept of declarative networking [15].Through it, network protocols and services were concisely specified and the complexity of explaining traditional language logic was avoided. In their proposed methodology, the language for declarative networking was Network Datalog where there are rules and predicates over fields and functions.Two extensions in the NDlog language was mentioned which are link restricted rules limiting expressiveness of language and soft storage model where storage data have associated lifetime. Boon Thau Loo et al also proposed pipelined semi naive evaluation where new tuples that are generated from the rules, as well as tuples received from other nodes, are used immediately to compute new tuples without waiting for the current (local) iteration to complete.It was also said that changes were allowed to happen during query processing. The assumption was that after a burst of changes, the network eventually quiesces (does not change) for a time long enough to allow all the queries in the system to reach a fixpoint. However, it did not provide debugging facilities on the networks architectures using declarative networking.

## 3.2 Distributed System Debugging

Nowadays Distributed Systems are large and finding faults in such large system is hard. Distributed nature of those faults makes it complex to identify. Again those faults are often partial, irregular and may result in abnormal behavior rather than system failure. So diagnosing a problem in such systems require collecting relevant information from many different nodes and correlating those with the problem.

One of the main sources of information that can give the idea about root cause of problems in a Distributed Systems is communication information between nodes. Looking at communication between different devices in Distributed System, it is seen that those are operated using various protocols. Among the protocols, Border Gateway Protocol (BGP) is quite popular. BGP is a standardized exterior gateway protocol for exchanging routing and reachability information among Autonomous Systems (AS) on the Internet. It is often classified as a path vector protocol and sometimes as a distance-vector routing protocol. In BGP protocol, updates and instabilities are complex. For this reason Anja Feldmann et al proposed a methodology to find instabilities visible in BGP routing that changes along three dimensions: time, prefix, and view [16]. It was mentioned that instability, causing BGP updates to propagate, can be observed at various monitoring points throughout the network. These updates were used to identify the instability origins. According to Anja Feldmann et al [16], specific sequence of BGP updates, which can be observed at particular points on the Internet by monitoring sessions, differ according to

- Location of the observation point
- Policies of the ASes along the AS path
- Effects of timing imposed on the message ordering
- AS topology

If the instability cause is an event internal to a given AS, the cause is located in the given AS. However, if the cause of the instability is due to an event at an external BGP session between two given AS, the cause is located at the edge between the two given AS. Although it gave the idea about routing instabilities, it can not provide information if routing instabilities did not happen.

Devices in Distributed Systems communicate to achieve a common goal. These devices, involved in communication, use communication system comprising mostly of switches, routers and nodes or physical devices. To diagnose any problem in these complex networks comprising of these devices is time consuming and erroneous. There are many works on this like analyzing the header content of packets [17] or analyzing control plane of switches to solve the problems. However Haohui Mai et al proposed analyzing data plane to debug the network problems [18] . In their proposed prototype Anteater, Forwarding Information Bases (FIBs) of the network topology and devices are collected and represented as boolean functions. The network operator specifies an invariant (like Loop-free forwarding,consistency or connectivity) to be checked. Anteater combines the specified invariant with the data plane state into instances of boolean satisfiability problem (SAT), thus performs analysis on it. If network state violates an invariant, a specific counterexample like a packet header, FIB entries, and path triggering the potential bug was provided. Anteater was applied to a large university campus network, 23 bugs were found and of those only 5 were false positives. However here facilities for providing information in cases like why an invariant did not violate network state was not mentioned.

Again in a Distributed System, different nodes communicating with each other are located on top of internet at different locations. For example, in peer to peer networks and client server applications, nodes run on top of the Internet and can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links in the underlying network. This type of

computer network built on top of another network is known as an overlay network. Among various works on overlay network were on how to make it reusable was one of the great challenges. To solve this, Boon Thau Loo et al proposed a system P2 to express overlay networks in a form that is highly compact and reusable [10]. The proposed system greatly simplified the process of implementing and deploying, as it used declarative logic language for specifying overlay. However, fault and security vulnerability detection was not ensured in the P2 system. Later Atul Singh et al extended existing P2 system so that queries can be processed for detecting faults, anomalies, and potential security vulnerabilities [19]. In the implemented system, every node has the tracer, which collects tuple exchange through every instrumented dataflow arcs. In those nodes, dataflows can be tapped.Tapping dataflow arcs are associated with high level query rules that generate dataflow graph. Planner performs tapping dataflow arc during generating graph. In order to capture information at the execution level, the planner must trace tuples entering a rule strand. Atul Singh et al [19] assumed here that strands of rule run sequentially from beginning to end. That is a rule strand will not have a new tuple until all tuples that are in flight within the rule strand have exited or have been dropped. The tracer can capture all executions of a particular rule on each node, and make those captured traces available for distributed querying. Although through proposed methodology [19], any fault or security vulnerabilities can be detected but it did not mention the explanation for events where vulnerabilities cannot be detected.

## 3.3 Data Provenance

Currently, the world is moving on data. Data is stored on a single physical device or distributed over a number of devices.When required, data are retrieved from these devices and are shown. However, there is a risk of data being erroneous, corrupted

or being manipulated at any time. Without integrity and validity of data. it will only mislead and give an unwanted result. So provenance or tracking data items from when it was inserted in physical device up to getting the view of data are needed to prove integrity and validity of data. To provide proof of data integrity and validity a number of works are being done [20] [21] [22] [23] [24]. There are also works [25] for interactive and user friendly display of provenance graph for better understanding.

In electronic devices whenever anyone requests data, devices retrieves and displays the data.For a given view of data items, identifying the base set of data items that exactly produce the view data items is termed the view data lineage problem. Yingwei Cui et al introduced lineage tracking to identify the source set of data items that produced the data on view [21] [26].It was shown view definitions can be transformed into a canonical form to generate tracing queries.After recursively applying the proposed algorithm in tracing queries, origin or existence of data can be shown.This helped in tracing interested data as well as to find cause of errors in scientific databases and network monitoring systems.Provenance also uses this property of tracing to find out reasons behind anything that happened. However explanation of why data items do not exist cannot be found from this literature study.

Although tracing the lineage of data shows proof of the existence of displayed output but it did not specify what storage data contributed each part of the shown output. Peter Buneman et al pointed this out and made a distinction between why the output exists and where each part of output comes from [27]. To make this distinction, a deterministic data model and a DQL(deterministic query language) were proposed. Using the proposed data model and query language, a specific collection of data in storage that produced specific parts of outputs can be verified and traced. Thus each part of output where it is located and why it appears can be shown. Although it was great to know about the contribution of different parts in storage in the displayed data, it failed to reason about why something was not displayed. Again, in order to

explain tuples that went missing from user query, [22] research was made on refined query for which they would have appeared or in cases like specifying what desired output needs obtained from a query, [28] there is work on suggesting what needs to be in the input.

## 3.4 Data oriented workflow Provenance

When various actions take place on input data to produce the desired output, a transformation of data takes place. If the transformation of data sets are visualized, it can be said to be data oriented workflow graph. In Data oriented workflow graph, nodes indicate transformation of datasets and edges indicates the flow of data input to an output from the transformations. These workflows are common in cases like scientific data processing and information extraction and there are many works on them [29].

Among Data oriented workflows, Generalized Map and Reduce Workflows (GMRWs) is a special case, where all transformations are either map or reduce functions. Among GMRWs, Hadoop is worth mentionable. Hadoop is an open source software framework for distributed storage and processing of dataset of big data. To show provenance in transformed data of GMRWs, Robert Ikeda et al implemented a prototype as an extension of Hadoop [30]. In the implemented prototype RAMP(Reduce And Map Provenance) extension, given sets of input data are processed by an acyclic graph of the map and reduce functions to produce output results. To each map output element, RAMP adds a unique id to the input element that generated the output element. For reduce functions, RAMP stores the reduce provenance as a mapping from a unique id for each output element to the grouping key that produced that produced the output. After map and reduce functions, RAMP can trace provenance by using an output element id. With output element id, RAMP accesses the reduce

provenance to determine the corresponding grouping key. After getting grouping key id, RAMP accesses the map provenance to retrieve all input elements id that is related.Thus tracing back of output is ensured. This helped to explain provenance in data orientated workflow like Hadoop however it was not generalized and fails to explain reasons if Hadoop did not do a map or reduce functions.

## 3.5 Provenance In Distributed Systems

In Distributed Systems, state of the system continuously changes. So any fault from any state that propagates with time is hard to track among the continuous changes of the system. To reason about changes in a Distributed System's state, Wenchao Zhou et al proposed a novel provenance model called Distributed Time-aware Provenance (*DTaP*) to aid debugging by explicitly representing time, distributed state, and state changes [5]. For this reason *DTaP* needs to continuously process, update and synchronize. Changes or provenance can be stored proactively or reactively. In the proactive approach, provenance information is logged, so it can quickly answer any query made. However in case of reactive approach, only non-deterministic events (such as received messages) are logged, so it requires time to extract provenance information and provide the response to queries. The choice of proactive or reactive provenance depends upon the underlying protocol, workload and also the metric (that is storage or latency of query) that the administrator of the Distributed System would like to optimize. However DTaP only cares for events that happened and for which provenance information was collected. So it fails to reason about events that did not happen.

Provenance can be used to explain reasons behind security breaches by tracking causality [31] [32] [33]. However in adversarial situation, nodes of the system might be manipulated and may contribute in a faulty or misleading provenance. So to ensure

finding faulty and misleading nodes even in adversarial situation Wenchao Zhou et al proposed secure network provenance [3]. In the proposed methodology, messages sent from nodes to nodes can be authenticated, each received message itself is evidence of its own transmission. The nodes attach some additional information like an explanation for the transmission of message. Thus, when a provenance query is issued on a correct node, that node can collect some evidence, such as messages it has locally received, or messages that are collected from other nodes. This evidence can be used to construct an approximation of provenance graph and queries can be answered.Although the problem of faulty node can lie about its local inputs and faulty nodes can equivocate remains. In the implemented prototype *SNooPy* there was no built-in redundancy. If any adversary destroys one of its nodes to destroy all the provenance state on it, the provenance graph that was formed from these parts may no longer be reachable through queries. However, *SNooPy* does not provide negative provenance, that is it can only explain the existence of a tuple (or its appearance or disappearance), but not its absence.

Exchange of provenance information might cause overhead and increase network latency.It is an active matter of concern in network provenance and many there are many works on it [34]. Wenchao Zhou et al implemented ExSpan*(EXtenSible Provenance Aware Networked systems)* [4] prototype to significantly reduce communication overhead by distributing provenance information. It was done by appending short provenance pointers to the tuples to identify the node maintaining relevant provenance information. After stimulation and experimentation, Wenchao Zhou et al showed implemented techniques imposed substantially less communication overhead. As for example, when providing provenance information for the path vector routing protocol, ExSPAN showed one ninth the communication overhead as incurred using traditional value-based distributed provenance techniques. Wenchao Zhou et al after ExSpan implemented NetTrails [35] declarative platform for incrementally maintaining provenance and interactively visualizing and navigating by querying network

provenance in a Distributed System. Like *SNooPy* [3] and *DTap* [5], *DTaP* cannot explain negative provenance.

## 3.6 Negative Provenance

Although provenance gathered from any system can give us proof what has happened by tracking, but it fails to reason about why anything did not happen. To solve this Yang Wu et al introduced negative provenance,where absence of events were explained in Distributed Systems [2]. It uses counterfactual reasoning to identify the conditions for which the missing event could have been realized. In positive events, there is way to generate a backtrace if any event occurs or some new state is generated.After a query is made, the system recursively inquires any event's related causes, until it reaches the set of root events that cannot be explained further. In case of negative events, there is a way to construct a similar backtrace like finding all the ways where a missing event could have happened.However complexity in negative event is greater, as positive provenance has specific chain of events that leads to an observed event, and negative provenance has to consider all possible chains of events causing observed event. Experiment on negative provenance was used in Software Defined Networking (SDN) debugging and BGP debugging on small scale deployment in virtual environment. So further researches on different cases as well as in negative provenance is required.

In SDN, networks can now be controlled by programs. Like all other programs, controller programs in SDN can have bugs. In network systems, existing tools provide support in debugging, however operator must fix the problem by himself. Yang Wu et al first proposed automated repair of SDN applications by using the key concept of data provenance. Provenance of data consists of tuples from which it was derived, and applying this idea recursively it is possible to trace the provenance of a tuple in

the output of a query to base tuple which explains how tuple came into existence. In previous works on provenance *SNP* [3] where provenance was considered in terms of packets and configuration data and SDN controller program was considered immutable. Yang Wu et al extended provenance to both network and data by treating controller program as another kind of data and using counter-factual reasoning [2]. In the proposed methodology, meta provenance considers program as a kind of data. So using provenance and backtracking in cases of bugs, one can find the root cause and know which part of the program contributed to the bug. However there may be infinitely many candidate repairs are there, so it explores the candidates in cost order makes it efficient.This paper mentioned that meta provenance does not consider the side-effects of a candidate repairs, it only considers whether the repair would solve the problem at hand. In order to validate the repair, the candidates are back tested using historical information from the network like a NetFlow trace or a sample of packets, along with some statistics, such as throughput and network latency, from the various end-hosts. So negative provenance helped to debug SDN program in this work. However, still it was in some cases and further research is required.

## 3.7 Summary

Existing works,such as SNP [3], ExSPAN [4] provides backtrace in case of any event. However, they failed to explain an why an event did not happen. Existing *Y!* [2] solved this problem by examining all possible causes that could have produced the missing effects. So the option of exploring the debugging process in network and system events through negative provenance is opened and it needs further research whether it can certainly contribute greatly in debugging process.

# Chapter 4

# NQ : A tool for Querying in Distributed System Positive and Negative Events through Provenance

Existing works on provenance [5] shows Distributed System can be queried for events that happened.There is also work on ensuring the authenticity of provenance information of nodes [3] as well as reducing provenance overhead [4].However, events that did not happen [2] was researched and found it be useful. However, it was tested in some cases and the result was found effective. So a methodology for a good querying and visualizing both positive and negative events are proposed. The aim of this method is to effectively query and visualize positive and negative events.

## 4.1 Overview of the proposed Querying Technique

The internal logical structure of the model along with algorithms associated in each of its components is elaborated here. To query events in Distributed Systems, the approach similar to *Y!* [2] is adapted to query events. This is done because *Y!* can answer queries for both positive and negative events by using provenance and counter factual reasoning. Visualization is also provided so that queried events can be better understood.

From the literature study, *Y!* was tested in some cases using network datalog. *Y!* used rapidnet [1], a toolkit for performing their experiments. *Y!*[2] used trema [36] and [37]frenetic for SDN applications, but these applications programs were translated into network datalog and was run on rapidnet. Since this work does not use translator, so it was omitted. The overview of the system structure to enable querying both positive and negative events is given below:

## 4.2 Description of System Architecture

The components of system architecture along with their responsibilities and performed actions are detailed in the following subsections.

### Protocol and invariant specification

Rapidnet [1] declarative networking engine uses NDlog as the language of declarative networking. Using it virtual environment can be created for experimenting with specified protocols and evolving network architecture designs. Initially, the protocols will be specified in NDlog, rapidnet will perform dataflow compilation and implemented

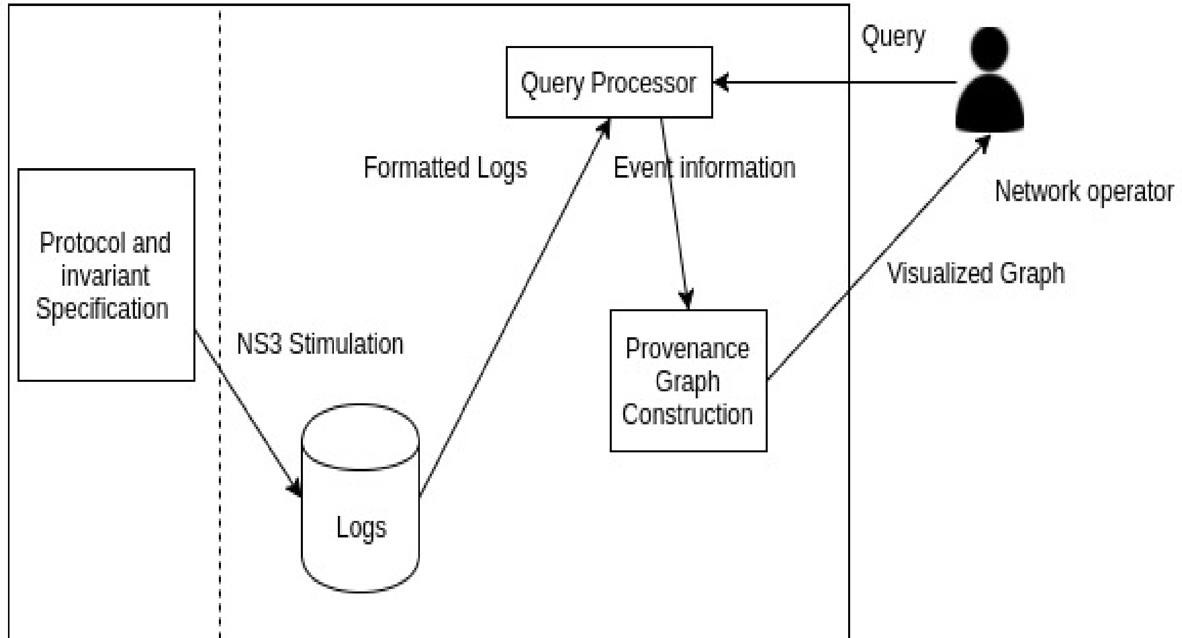


Figure 4.1: Architecture of the NQ

protocols will be ready for use in the ns3 program. After getting the protocols ready,a ns3 program along with rapidnet libraries is written mentioning invariants like a number of nodes or switches or routers, assigning the IP address, when links between nodes will be shut down and others. Then the program will be executed and simulations will be done.

Among the protocols, experiment was done with BGP Protocol.In the Figure 4.3, materialize(link, infinity, infinity, keys(1,2)) statement specifies the relation link, lifetime of each tuple in the relation as infinity, maximum size of the relation as infinity, and fields making up the primary key of relation link. The first rule r1 implies that tuple path at node X will be generated if there is another tuple link at node X, and will have fields Y and C same, but P formed from the concatenation of X and Y. Other rules can similarly be explained. After specifying these rules of how nodes

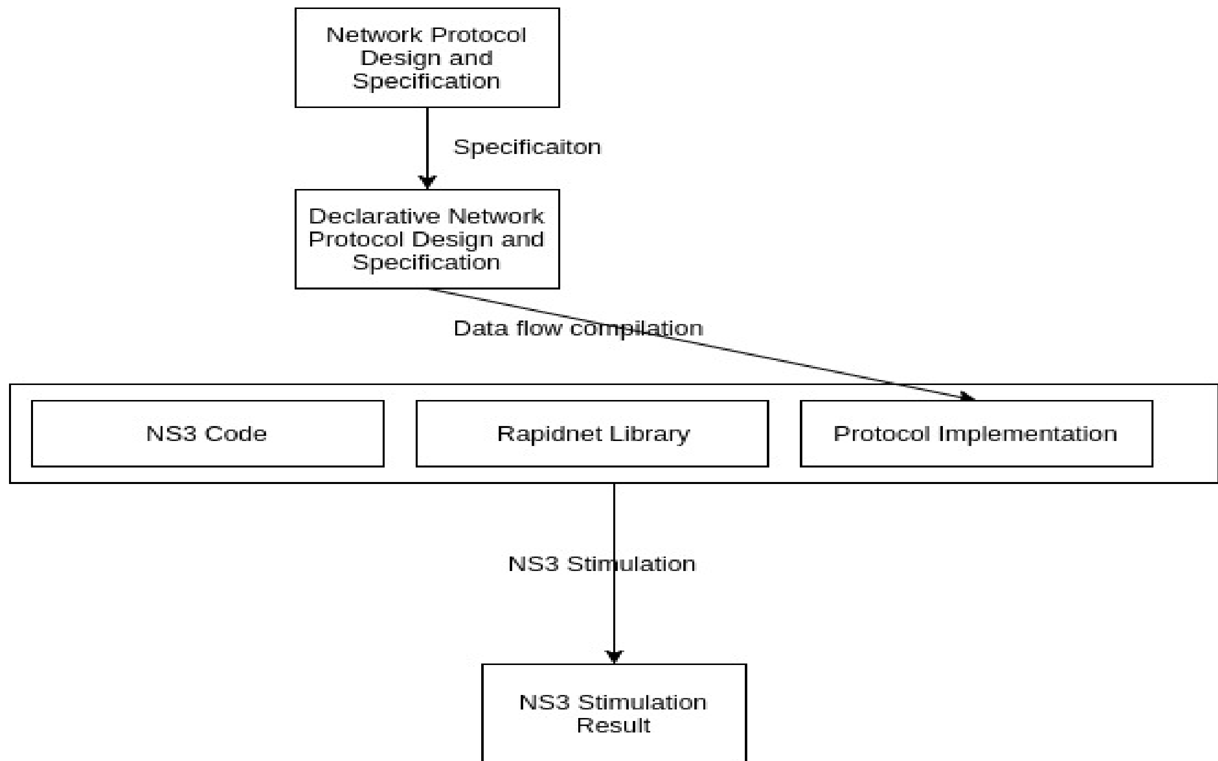


Figure 4.2: NS3 stimulation with specified protocols

will communicate, ns3 programs were written using the rapidnet library to perform simulations.

## Logs and formatting

After running ns3 program, simulation will create the virtual environment where specified interactions between nodes will take place. During simulation occurred events are captured explicitly by rapidnet and system log can be obtained. Using the rapidnet, the libpcap facility was enabled to record packet traces in virtual environment. In this way logs required for querying events and constructing provenance graph are collected and these are then needed to be formatted so that query operations can be done on them.

```
materialize(link,infinity,infinity,keys(1,2)).
materialize(path,infinity,infinity,keys(4:list)).
materialize(bestPath,infinity,infinity,keys(2)).

/* Rules */

r1 path(@X,Y,C,P) :- link(@X,Y,C),
                 P1:=f_append(X),
                 P2:=f_append(Y),
                 P:=f_concat(P1,P2).

r2 path(@X,Y,C,P) :- link(@X,Z,C1),
                bestPath(@Z,Y,C2,P2),
                C:=C1+C2,
                f_member(P2,X)==0,
                P1:=f_append(X),
                P:=f_concat(P1,P2).

r3 bestPath(@X,Y,a_MIN<C>, P) :- path(@X,Y,C,P).
```

Figure 4.3: BGP datalog used in case study

Here it is assumed that log will be formatted in the structure ($\pm\tau$, N, t, r, c), where

- $\pm\tau$ indicates tuple was derived($+\tau$) or underived($-\tau$).
- N indicates the node on which the event took place.
- t indicates the time of occurrence of the event.
- r indicates the rule. Since tuples are derived using rules, so these rules are used to trace the tuples origin and which tuples contributed to the formation of a derived tuple.
- c indicates derivation counter. Derivation counter is used if same tuple is derived differently at different times. Like path tuple from a node to another node may change because of link failure. So resultant derived tuple will be different. For tuples appearing for the first time, deriver counter 1 is set and incremented by

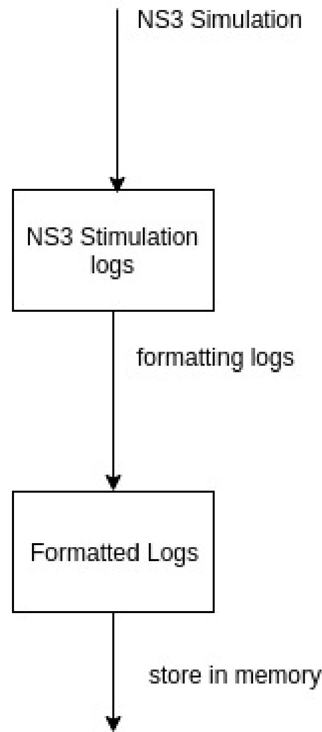


Figure 4.4: Getting Logs used in case study

1 for further derivations.

After formatting the logs these are now sent to query processor to answer queries made by a network operator.Here, logs are kept in the main memory without storing them in hard disks.

## Query Processor

After a network operator issues query about a tuple or packet, the query and previously collected formatted logs are processed and output events are obtained. These events are used in the construction of provenance graph.Figure 4.5 gives an overview of how events are formed from a query. From network operators inquiry about a tuple or packet, these query is then searched among logs which are formatted in the

mentioned format. After these resultant events are obtained and send for further querying and graph construction.

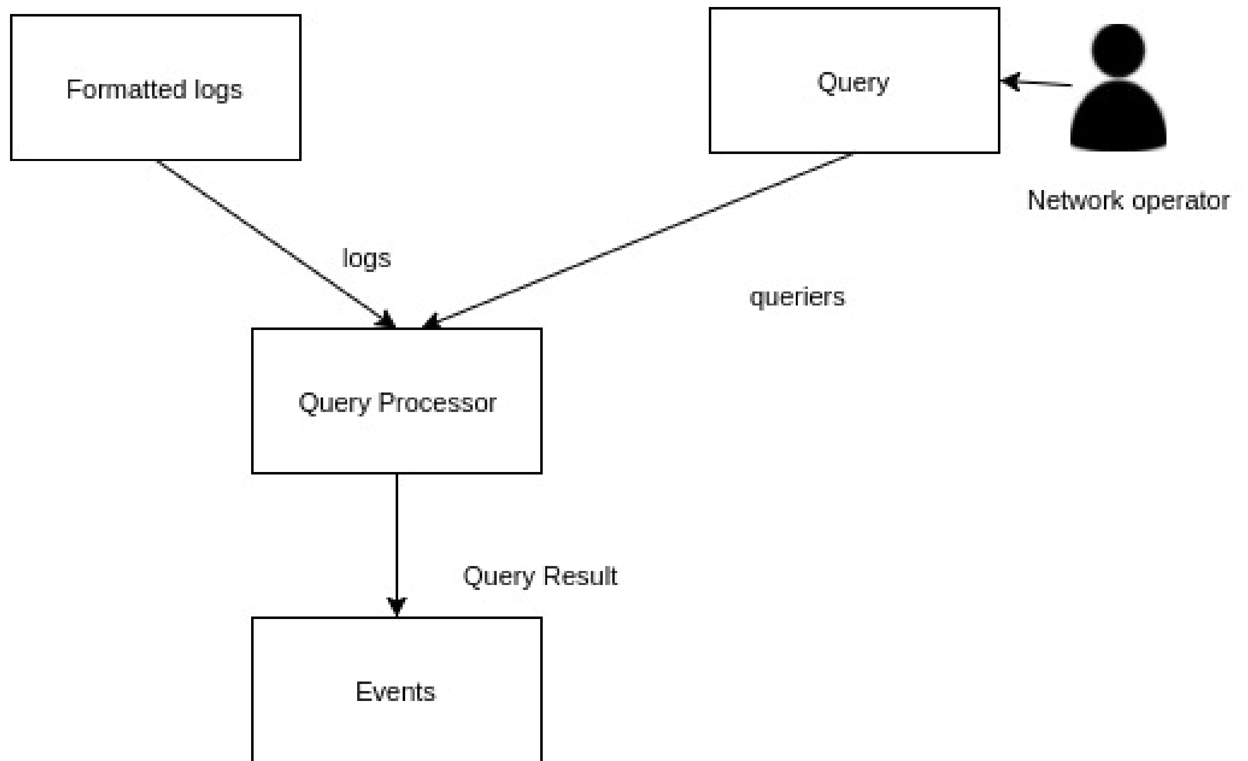


Figure 4.5: Processing queries

## Provenance Graph

The alogrithms for construction of provenance consists of followings:

**Algorithm 1** Queries

**ExistQuery** $(EXIST([t1, t2], N, \tau))$

$S \leftarrow \emptyset$

**foreach** *(+τ, N, t, r, c)* $\in$ *Log* : $t1 < t < t2$ **do**

$S \leftarrow S \cup \{APPEAR(t, N, \tau, r, c)\}$

**end**

**foreach** *(-τ, N, t, r, c)* $\in$ *Log* : $t1 < t < t2$ **do**

$S \leftarrow S \cup \{DISAPPEAR(t, N, \tau, r, c)\}$

**end**

**return** $S$

**AppearQuery** $(APPEAR(t, N, \tau, r, c))$

**if** $BaseTuple(t)$ **then**

**return** $(INSERT(t, N, \tau))$

**else if** $LocalTuple(N, )$ **then**

**return**$(DERIVE(t, N, \tau, r))$

**else**

**return**$(RECEIVE(t, N \leftarrow N'.N, \tau)$

**InsertQuery** $(INSERT(t, N, \tau))$

**return** $\emptyset$

**DeriveQuery** $(DERIVE(t, N, \tau, \tau : -\tau 1, \tau 2....))$

$S \leftarrow \emptyset$

**foreach** $\tau i$ **do**

  **if** $(+\tau,\ N,\ t,\ r,\ c) \in Log$ **then**

    $S \leftarrow S \cup \{APPEAR(t, N, \tau i, r, c)\}$

  **else**

    $tx \leftarrow max\ \ t_0 < t : (+\tau, N, t0, r, 1) \in Log$

    $S \leftarrow S \cup EXIST([tx, t], N, i, c)$

**return** $S$

---

**ReceiveQuery** $(RECEIVE(t, N1 \leftarrow N2, +\tau))$

$ts \leftarrow max\ \ t_0 < t : (t, N1, N2, +\tau) \in Log$

**return** $\{SEND(ts, N1 \rightarrow N2, +\tau), DELAY(ts, N2 \rightarrow N1, +\tau, t - ts)\}$

---

**SendQuery** $(SEND(t, N \rightarrow N0, +\tau))$

$FIND(+\tau, N, t, r, c) \in Log$

**return** $\{APPEAR(t, N, \tau, r)\}$

---

**NExistQuery** $(NEXIST([t1, t2], N, \tau))$

**if** $\exists t < t1 : (-\tau, N, t, r, 1) \in Log$ **then**

  $tx \leftarrow max\ \ t_0 < t : (-\tau, N, t0, r, 1) \in Log$

  **return** $\{DISAPPEAR(tx, N, \tau), NAPPEAR((tx, t2], N, \tau)\}$

**else**

  **return** $\{NAPPEAR((0, t2], N, \tau)\}$

**NDeriveQuery** $(NDERIVE([t1, t2], N, \tau, r))$

**foreach** $\tau i$ **do**
  $S \leftarrow S \cup \{NEXIST([t1, t2], N, \tau i)\}$
  **return** $\{S\}$

**NSendQuery** $(NSEND([t1, t2], N, +\tau))$

**if** $\exists t1 < t < t2 : (-\tau, N, t, r, 1) \in Log$ **then**
  **return** $\{EXIST([t1, t], N, \tau), NAPPEAR((t, t2], N, )\}$
**else**
  **return** $\{NAPPEAR([t1, t2], N, \tau)\}$

**NAppearQuery** $(NAPPEAR([t1, t2], N, \tau))$

**if** $BaseTuple(t)$ **then**
  **return** $\{NINSERT([t1, t2], N, \tau))\}$
**else if** $LocalTuple(N, \tau)$ **then**
  **return**$\{NDERIVE([t1, t2], N, \tau, r)\}$
**else**
  **return**$\{NRECEIVE([t1, t2], N, +\tau)\}$

**NReceiveQuery** $(NRECEIVE([t1, t2], N, +\tau))$

$S \leftarrow \emptyset, t_0 \leftarrow t1 - \Delta max$
**foreach** $N_i \in SENDERS(\tau, N)i$ **do**
  $X \leftarrow \{t_0 \leq t \geq t_2 | (+\tau, N_0, t, r, 1)2 \in log\}$ $t_{\mathrm{x}0}$ **for** $i \leftarrow 0$ **to** $i \leq |X|$ 1 **do**
    $S \leftarrow \{NSEND((t_{\mathrm{x}}, Xi), N_0, +\tau), NARRIVE((t1, t2), N_0 \rightarrow N, X_{\mathrm{i}}, +\tau)\}$
    $t_{\mathrm{x}} \leftarrow X_{\mathrm{i}}$
  $S \leftarrow S \cup \{NSEND([t_{\mathrm{x}}, t_2], N_0, +\tau)\}$
**return** $S$

**NArriveQuery** $(NARRIVE([t_1, t_2], N_1 \rightarrow N_2, t_0, +\tau))$

**FIND**$(+\tau, N_1, t_0, r, 1) \in Log$

**return** $\{SEND(t_0, N_1 \rightarrow N_2, +\tau), DELAY(t0, N_{12}, +\tau, t_2 - t0)\}$

In the proposed system NQ, provenance was represented as a Directed Acyclic Graph (DAG) where the vertices are considered events and the edges indicate the direct causal relationships between the vertices. A top-down method was taken for getting the provenance graph. Here a function was defined as getProvenanceGraph(v), where v represents a vertice or event. In the function getProvenanceGraph(v), events are queried so that responsible event for the derivation of v is found which are further recursively queried using the getProvenanceGraph(v). In this way, until the leaf or base events are found, the query continues. In the proposed system, followings are the basic vertices for positive provenance,

- EXIST([t1, t2], N, $\tau$ ): Tuple $\tau$ existed on node N from time t1 to t2.
- INSERT(t, N, $\tau$ ), DELETE(t, N, $\tau$ ): Base tuple $\tau$ was inserted (deleted) on node N at time t.
- DERIVE(t, N, $\tau$ ), UNDERIVE(t, N, $\tau$ ): Derived tuple $\tau$ acquired (lost) support on N at time t.
- APPEAR(t, N, $\tau$ ), DISAPPEAR(t, N, $\tau$ ): Tuple $\tau$ appeared or disappeared on node N at time t.
- SEND(t, N $\rightarrow$N$\theta$, $\pm\tau$ ), RECEIVE(t, N $\leftarrow$N$\theta$, $\pm\tau$): $\pm\tau$ was sent (received) by node N to/from N$\theta$ at t.
- DELAY(N $\rightarrow$ N$\theta$, $\pm\tau$, d): $\pm\tau$, sent from node N to N$\theta$ took time d to arrive.

In the above basic vertices for positive provenance, the events INSERT/DELETE, APPEAR/DISAPPEAR, and DERIVE/UNDERIVE are considered symmetric. So these are not mentioned here. In case of Negative provenance, followings are are the basic vertices that are considered in proposed system :

- NEXIST([t1, t2], N, $\tau$ ): Tuple $\tau$ never existed on node N in time interval [t1, t2].
- NINSERT([t1, t2], N, $\tau$ ), NDELETE([t1, t2], N, $\tau$ ): Tuple $\tau$ was never inserted (removed) on N in [t1, t2].
- NDERIVE([t1, t2], N, $\tau$ ), NUNDERIVE([t1, t2], N, $\tau$ ): $\tau$ was never derived (underived) on N in [t1, t2].
- NAPPEAR([t1,t2],N, $\tau$ ), NDISAPPEAR([t1,t2],N, $\tau$ ): Tuple never (dis)appeared on N in [t1, t2].
- NSEND([t1, t2], N, $\tau$ ), NRECEIVE([t1, t2], N, $\tau$ ): was never sent (received) by node N in [t1, t2].
- NARRIVE([t1, t2], N1 $\rightarrow$ N2, t3, $\tau$ ): was sent from N1 to N2 at t3 but did not arrive within [t1, t2].

In the above, negative vertices differ in time interval as positive vertices have a particular time of occurrence whereas negative events do not. So the time during which negative vertices are not present is represented as the time interval.All the above functions are clearly specified except the only function Sender that is used in NReceive is made such that it queries the node about its connection with neighbouring nodes and the tuple properties so that possible sender nodes are obtained. Again the provenance graph constructed from these basic vertices is somehow complex, so the concept of supervertices is applied. NExist, NDerive and NAppear vertice practically

means that a tuple is absent and for this reason thesee are combined to form the supervertices Absence([t1, t2], N, $\tau$ ). Similarly for positive vertices, Exist, Derive and Appear, Existence([t1, t2], N, $\tau$) is used.

## Visualization

After obtaining provenance graph, Java JGraph library was used for the construction of a tree based display of provenance graph. The root of the provenance graph is the queried tuple's existence or absence and the vertices that are gradually added explains the reason behind why a tuple exists or why it was absent. Figure 4.6 explains the process of visualization.

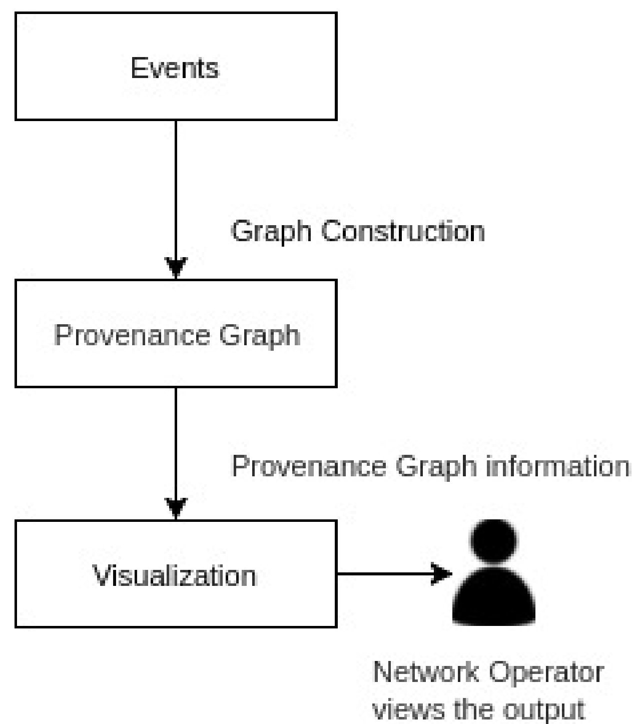


Figure 4.6: Visual Display of provenance graph

## 4.3 Summary

In this chapter, the methodology was described step by step for a better understanding of the architecture and how it is made. In the proposed methodology, algorithms of existing *Y!* are used. However, *Y!* did not give the algorithm for some functions like Sender (which searches probable senders in case of events that did not happen because of another node not sending information), super vertices from basic vertices and some functions were modified according to understanding like NDerive in order to make the tool.

# Chapter 5

# Implementation and Result Analysis

From the literature study, it is seen that negative provenance is recently added in the field of diagnosis of faults of Distributed Systems and the previous chapter described the mechanism of achieving this. In this chapter, a description of the platform and environmental setup to execute the research experiments and analyze performance are given. The goal is to use different test cases and find the effectiveness of the used system to identify both positive and negative event in Distributed Systems in the virtual environment. The proposed method was applied of 5 test cases of BGP protocol, of them 3 are negative and 2 are positive events. The output obtained was analyzed with respect to the amount of storage used, the speed of query reply and responded vertices of provenance graph. In all cases, the proposed method ensured effectiveness and readability. The output was displayed in provenance graph for easy understandability of the network operators.

## 5.1 Implementation Details

In this section, appliances that are required for proposed system implementation and analysis are discussed. The architecture of NQ consists of rapidnet declarative networking engine with ns3 network simulator, java, and jgraph. The rapidnet makes use of network datalog, declarative networking language to specify protocols and communicate between nodes.

(i) **NS3:** NS3 is the network simulator version 3 where discrete events in networks are stimulated in a virtual environment. It is primarily used for educational and research purpose to meet the modern networking simulation needs. It also facilitates real time scheduler to enable interacting with system [11]. and evaluated.

(ii) **Rapidnet:** For quick simulation, implementation and experimentation of network protocols, a toolkit which integrated declarative networking engine with NS3 is Rapidnet [1]. It uses declarative networking language NDlog [15] to specify the behavior of the system. Rapidnet is a development toolkit for rapid simulation, implementation, and experimentation of network protocols. Rapidnet utilizes declarative networking, a declarative, database-inspired extensible infrastructure that uses query languages to specify behavior. Rapidnet integrates a declarative networking engine with the emerging ns-3 network simulator.

(iii) **Eclipse:** Eclipse is the most widely used Java integrated development environment. It is primarily used for developing Java applications but can also be used for developing other applications. It was used in this current work for parsing log files and getting the building the user interface [38].

(iv) **JGraph:** Jgraph is a java library for swing diagramming. It provides user friendly interaction as well as visualization of node edge graphs. It is used as

workflow editor, a business process modeling tool, UML tool, electronic circuit diagrammar. network/telecoms visualization and others [39].

(v) **System configuration:** The configuration that is used in the experiment is as follows

  (a) **Processor :** Intel(R) Core(TM) i3 -4130U CPU @3.40 GHz

  (b) **RAM :** 4GB

  (c) **Operating System :** Ubuntu 14.04 LTS

  (d) **Operating System Type :** 64-bit Operating System

## 5.2 Experimental Datasets

In order to verify the effectiveness of whether or not NQ can answer questions about both positive and negative symptoms. Here Case study is focused on BGP and the work is based on NDlog encoding of general path vector protocol. 6 cases of BGP routing is used as test cases and report on them are filed. In these BGP cases, it was considered connection between them exists if notify each other at a time rather than periodically. These cases are as follows:

(i) **Off path change** In the first query, AS2 had the bestroute from AS2 to AS7 through AS 1,3,4,5 and 6 respectively. But it loses this bestroute due to the addition of another route between 8 and 9.In Figure 5.1 we can see the topology used and query (Q1 in Table 5.1) was operated based on this topology.

(ii) **Blackhole** A router advertises the route to certain hosts without actually having the route. This prevents getting queries from fake hosts. In Figure 5.2 AS10 advertises the route to AS11 and AS12 without actually having one. So any query (Q2 in Table 5.1) issued does not have activities on AS11 and AS12.

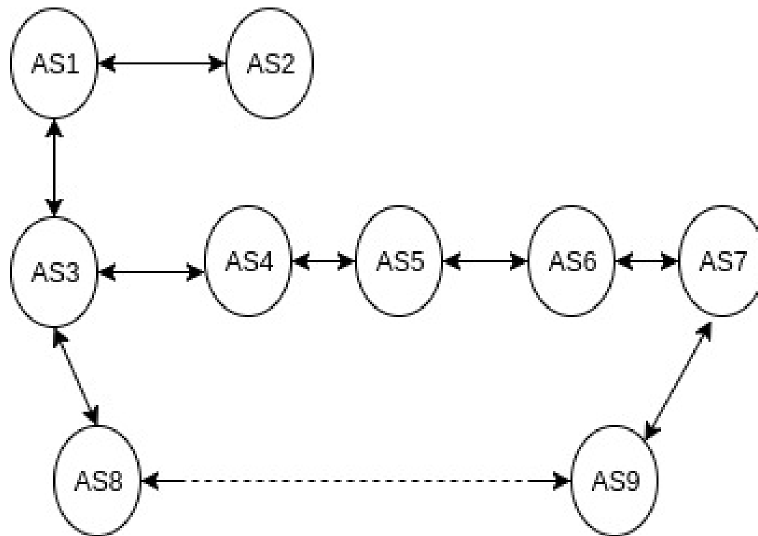


Figure 5.1: A BGP Topology used in off path change.

(iii) **Link Failure** In this case, a situation where ISP loses connectivity due to failure is considered. The topology used was same as Figure 5.1.The connection between 8 and 9 failed and query (Q3 in Table 5.1) was made to ask the reason.

(iv) **Never Existed** In this case, query (Q4 in Table 5.1) was asked to show why the path to a never existed router did not exist. The topology used was same as Figure 5.1.

(v) **Positive Provenance** Query 5 and 6 (in Table 5.1) asks the reason behind the existence of path and bestroute between AS.

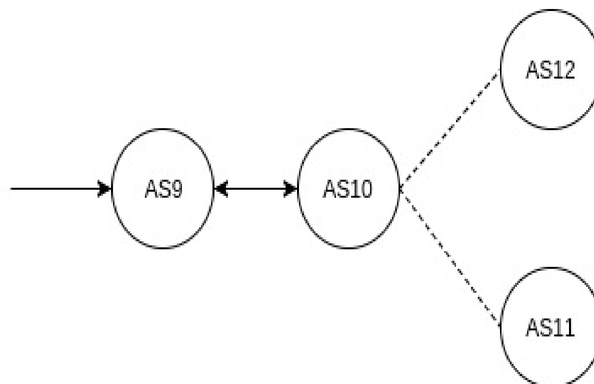


Figure 5.2: A BGP Topology used in BGP blackhole query.

| Test Cases | | |
|---|---|---|
| Query no | Type | Description |
| 1 | BGP | NExist([t1,t2], bestPath(@AS2, TO=AS7)) |
| 2 | BGP | NExist([t1,t2], packet(@AS7, SRC=AS2)) |
| 3 | BGP | NExist([t1,t2], bestPath(@AS6, TO = AS5)) |
| 4 | BGP | NExist([t1,t2], bestPath(@AS2, TO=AS7)) |
| 5 | BGP | Exist([t1,t2], bestPath(@AS1, TO=AS7)) |
| 6 | BGP | Exist([t1,t2], packet(@AS1, TO=AS10)) |

Table 5.1: Test Cases.

## 5.3 Result Analysis

Positive provenance or explanation of any event that happened was asked in Q5 and Q6. In Q5, how a bestPath existed was asked and then in Q6 how packet arrived at its destination was asked. The reply was given in detailed format containing reply of why it happened and the time required to reply was calculated by execution time of the function. In Q5, 64 vertices were in response and time required was .04 second on average and in Q6 72 vertices were in response and time taken was .06 on average.

The query of why bestPath (Q1) was not available at different times was asked to the NQ and the answer was given with detailed in the average of time .07 seconds, containing 35 vertices in response. Similarly, in other cases of negative provenance, the time required and vertices are as mentioned in Table 5.2. As the simulation ran, all the pcap files, generated by rapidnet [1] was collected and the memory consumption is as shown in Figure 5.3. It is seen that the memory consumed by generated pcap files increases greatly with the increase in number of nodes.

The summary of response vertices obtained from NQ for Q1, contained Absence of "bestPath" due to emergence of new path with shortest cost. In case of Q2, responses consisted of how the packet did not arrive at a node despite it was present and sent from another node as a node created black hole using false advertise. In Q3 the "bestPath" changed as the path between two nodes disappeared. In Q4 , the obvious answer would be a single absence event and it appeared in our tool. In Q5, best path between two vertices that was present between a time and reason it disappeared is show. In Query 6, it was shown how packet was sent from one cite to another using positive provenance.

After a query is asked, the relevant output is visualized using [39] JGraph. In the NQ tool, both basic vertices and super vertices options are kept so that querier can go in depth about the reason behind a positive or negative event. The tool takes the log file as input and produces the provenance graph. The positive provenance in the tool can be considered user friendly, however negative provenance needs exact input, so it is somewhat rigid.

| Result | | |
|---|---|---|
| Query no | Time | Vertices in response |
| 1 | .04 seconds | 35 |
| 2 | .07 seconds | 73 |
| 3 | .02 seconds | 7 |
| 4 | .01 seconds | 1 |
| 5 | .04 seconds | 64 |
| 6 | .07 seconds | 72 |

Table 5.2: Result Obtained.

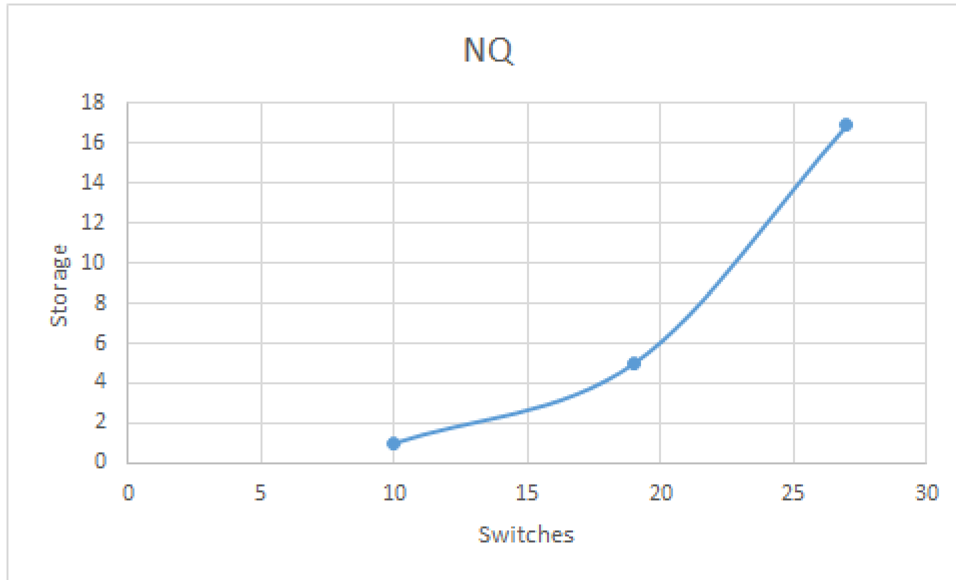


Figure 5.3: Disk Consumption

## 5.4 Comparative Analysis

In Q1, Q2 ,and Q3 the comparison according the existing *Y!* [2] is given in Table 5.3 and 5.4. According to the Table 5.3, it can be said that the response vertices are greater in number compared to existing *Y!*. Although 7 showed from Q3 achieved good response, and others have considerable high number of response vertices. So overall the performance of NQ to explain causes with number of vertices is lower than existing *Y!*. The main reason behind this can be considered as how they interpreted the rules behind derivation and the way the considered super vertices and pruning.

In Table 5.4, it can be seen that queries required comparatively less time than existing *Y!*. It is due to all logs are made available in primary memory rather than in hard disk. So the result obtained was faster than the existing *Y!*. Also from the Table 5.2, response time increases with the increase of number of vertices in response, that is as more vertices needs more explanation according to their rules they are derived from.

| Number of Vertices in response | | |
|---|---|---|
| Query no | Vertices in response | Existing *Y!* response |
| 1 | 35 | < 20 |
| 2 | 73 | < 15 |
| 3 | 7 | < 20 |

Table 5.3: Comparison of Vertices in Response.

| Time of response | | |
|---|---|---|
| Query no | Time | Existing *Y!* response time |
| 1 | .04 seconds | < .15 seconds |
| 2 | .07 seconds | < .05 seconds |
| 3 | .02 seconds | < .1 seconds |

Table 5.4: Comparison of Time.

## 5.5 Summary

NQ could detect the problems by querying and then return the responses like the existing *Y!*. Visualization made it user friendly and possible to analyze the situation more closely. However, in some cases like huge log data and efficient mechanism for storing them as well as the number of response vertices, existing *Y!* is good compared to NQ.

# Chapter 6

# Conclusion

In this report, the negative provenance in Distributed Systems have been explored and was applied on different test cases to ensure the effectiveness of the proposed system. The contributions of this work are using negative provenance in different test cases to prove its effectiveness. This chapter concludes the report by providing summary and direction of future work.

## 6.1 Discussion

Negative provenance to query different situations that are mentioned in this report can help to provide the diagnosis of different faults in the Distributed System. The tool developed NQ gives us the detailed description of reasons of fault and it was tested on some faults of well known BGP protocol. It is seen to perform good and compared to existing *Y!* its performance can be acceptable. As mentioned in the previous chapter, the number of vertices in response was found greater than existing *Y!*, it is because of different way of interpretation of rules. The time required to get the response can be considered good compared to existing approach as it used primary memory, that

is loading the entire log in main memory. However, in case of scalability existing system will have superiority over NQ. The visualized vertices of NQ tool which was not in existing *Y!* will certainly contribute greatly in interpretation. In NQ, showing options of both basic and super vertices are kept so that one can inquire in more detailed way and through visually.

In this report, using NQ 6 cases are analyzed. and of them 2 are positive and 4 are negative. The response obtained from NQ gave a detailed description of why any event happened or did not happen by tracing the available positive events from the log and matching them against NDlog rules. The visualization of the entire provenance graph enabled easier analysis of the result obtained from the query. So better diagnosis of the result can be obtained from visualization of provenance graph using NQ.

The entire tool with visualized provenance graph and implementation following the existing *Y!* can be considered the main achievement in this work. Query time is improved in this work, which however uses the fact that the entire log is kept in primary memory rather than disk storage. Although response vertices that give description about reasons behind an event are more compared to the existing *Y!*, however, this can play a great role in explaining the detailed version of an event.

All the results that are obtained using NQ depend on the interpretation of NDlog rules and their use for deriving the tuples that are responsible information of another tuple. So logical interpretation of NDlog rules and using them to query the events, the outputs obtained will certainly depend on that interpretation. Test cases were created using those rules, and NQ performed logical interpretation of these rules to produce the output. So results in test cases are certainly according to how the rules were interpreted.

## 6.2 Threats to validity

In this section threats that can hamper the accuracy and the effectiveness of the proposed model is discussed.These threats are as follows:

(i) This model assumes that system log is available and accessible. The available system log contains enough information and specifications of how input is processed to get output. Provenance graph is drawn based on information from the log. So without valid information and specifications result may vary

(ii) The implementation of NQ and experiments were done on Java programming language. So expected result obtained from the experimental setups may vary according to platform and technologies.

(iii) The implementation was completely based on log file obtained from rapidnet declarative networking engine. So because of any inaccuracy or fault of the system, results are affected.

(iv) It is assumed that the specifications and rules are clearly stated and derivation logic are transparent. So obtaining tuple and explaining reasons as shown in vertices depends on them. The result may be affected if derivation logic is considered in different ways.

## 6.3 Future Work

The necessity of accountable and secured applications and systems are highly important in this age of information. Provenance provides the ability to account a system for any event that happened and also through [2] the reason of why anything did not happen. The main use of provenance can be seen in complex Distributed Systems as

well as the modern cloud which provides services to clients but also should remain accountable to them. As with current progress in this work, it will be possible to apply the concept of negative provenance in different ways. However, after this, the main focus will be given on testing negative provenance on Hadoop [40] on real nodes.This will enable answering different errors in cases like why Hadoop map or reduce work has been relocated different node than previously assigned node, in cases like link failure or node shutdown and some similar other test cases.

# References

[1] Rapidnet [Access Date : 28-10-2017]. Url: `http://netdb.cis.upenn.edu/rapidnet/`.

[2] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. *ACM SIGCOMM Computer Communication Review - SIGCOMM'14*, 44(4):383–394, Aug 2014.

[3] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure network provenance. In *Proc. SOSP '11 Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 295–310, Oct 2011.

[4] 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 *Proc. 2010 ACM SIGMOD International Conference on Management of data*, pages 615–626, June 2010.

[5] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Distributed time-aware provenance. *Proceedings of the VLDB Endowment*, 6(2):49–60, December 2012.

[6] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo. Automated network repair with meta provenance. In *Proc. 14th ACM Workshop on Hot Topics in Networks*, November 2015.

[7] C. Wang, S. Chen, and J. Zic. A contract-based accountability service model. In *Proc. ICWS '09 Proceedings of the 2009 IEEE International Conference on Web Services*, pages 639–646, July 2009.

[8] D. Zhao, S. Chen, T. Malik, and I. Raicu. Distributed data provenance for large-scale data-intensive computing. In *Proc. 2013 IEEE International Conference on Cluster Computing*, 2013.

[9] Computer Network [Access Date : 28-10-2017]. Url: `https://web.archive.org/web/20120121061919/http://www.atis.org/glossary/definition.aspx?id=6555`.

[10] B. T. Loo, P. Maniatis, T. Condie, T. Roscoe, J. M. Hellerstein, and I. Stoica. Implementing declarative overlays. *ACM SIGOPS Operating Systems Review - SOSP '05*, 39(5):75–90, December 2005.

[11] NS3 [Access Date : 28-10-2017]. Url: `https://www.nsnam.org/`.

[12] Client-Server Systems [Access Date : 28-10-2017]. Url: `https://www.slideshare.net/gheethumariajoy/clientserver-systems`.

[13] N tier Architecture [Access Date : 28-10-2017]. Url: `https:https://stackify.com/n-tier-architecture/`.

[14] Note on Networking Model Client/Server P2P [Access Date : 28-10-2017]. Url: `https://www.kullabs.com/classes/subjects/units/lessons/notes/note-detail/3042`.

[15] 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 *Proc. 2006 ACM SIGMOD international conference on Management of data*, pages 97–108.

[16] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs. Locating internet routing instabilities. *ACM SIGCOMM Computer Communication Review*, 34(4):205–218, 2004.

[17] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proc. NSDI*, pages 9–9, 2012.

[18] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, B. P. Godfrey, and S. T. King. Debugging the data plane with anteater. *ACM SIGCOMM Computer Communication Review - SIGCOMM '11*, 41(4):290–301, 2011.

[19] A. Singh, T. Roscoe, P. Maniatis, and P. Druschel. Using queries for distributed monitoring and forensics. *ACM SIGOPS Operating Systems Review - Proceedings of the 2006 EuroSys conference*, 40(4):389–402, 2006.

[20] K.-K. Muniswamy-Reddy, D.A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proc. ATEC '06 Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 4–4, May 2006.

[21] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE '00 Proceedings of the 16th International Conference on Data Engineering*, March 2002.

[22] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *Proc. 2010 ACM SIGMOD International Conference on Management of data*, 2010.

[23] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. Orchestra: Facilitating collaborative data sharing. *ACM SIGMOD Record*, 37(3):26–32, 2007.

[24] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *Proc. 2010 ACM SIGMOD International Conference on Management of data*, pages 951–962, June 2010.

[25] P. Macko and M. Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *Proc. TaPP*, 2011.

[26] J Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. 32nd international conference on Very large data bases*, pages 1151–1154, January 2005.

[27] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In *Proc. ICDT '01 Proceedings of the 8th International Conference on Database Theory*, pages 316 – 330, 2001.

[28] A. Meliou and Suciu. D. Tiresias: The database oracle for how-to queries∗. In *Proc. SIGMOD*, page 337–348, May 2012.

[29] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Proc. 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350, June 2008.

[30] R. Ikeda, H. Park, and J. Widom. Provenance for generalized map and reduce workflows. In *Proc. 6th workshop on Workflows in support of large-scale science*, 2011.

[31] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *Proc. Network and Distributed System Security Symposium (NDSS)*, feb 2005.

[32] R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *ACM Transactions on Storage (TOS)*, 5(4), December 2009.

[33] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *Proc. 33rd international conference on Very large data bases*, pages 675–686, September 2007.

[34] M. Liu, N. E. Taylor, W. Zhou, Z. G. Ives, and B. T. Loo. Maintaining recursive views of regions and connectivity in networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1126–1141, August 2010.

[35] W. Zhou, Q. Fei, S. Sun, T. Tao, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Nettrails: A declarative platform for maintaining and querying provenance in distributed systems. In *Proc. 2011 ACM SIGMOD International Conference on Management of data*, pages 1323–1326, 2011.

[36] Trema [Access Date : 28-10-2017]. Url: `https://trema.github.io/trema/`.

[37] Frenetic [Access Date : 28-10-2017]. Url: `https://github.com/frenetic-lang/frenetic`.

[38] Eclipse Wiki [Access Date : 28-10-2017]. Url: `https://wiki.eclipse.org/Main_Page`.

[39] Jgraph [Access Date : 28-10-2017]. Url: `https://github.com/jgraph/jgraphx`.

[40] Hadoop [Access Date : 28-10-2017]. Url: `http://hadoop.apache.org/`.