MODELING PERFORMANCE OF INTERNET-BASED SERVICES USING CAUSAL REASONING

A Thesis
Presented to
The Academic Faculty

by

Muhammad Mukarram Bin Tariq

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science, College of Computing

Georgia Institute of Technology
May 2010

MODELING PERFORMANCE OF INTERNET-BASED SERVICES USING CAUSAL REASONING

Approved by:

Professor Mostafa Ammar,
Professor Nicholas Feamster,
Committee Co-Chairs
School of Computer Science,
College of Computing
Georgia Institute of Technology

Professor Mostafa Ammar,
Professor Nicholas Feamster, Advisors
School of Computer Science,
College of Computing
Georgia Institute of Technology

Professor Alexander Gray
Computational Science and
Engineering Division,
College of Computing
Georgia Institute of Technology

Dr. Walter Willinger
Information and Software Systems
Research Center
AT&T Labs Research

Professor Ellen Zegura
School of Computer Science,
College of Computing
Georgia Institute of Technology

Date Approved: 11 February 2006

To my family.

ACKNOWLEDGEMENTS

I am immensely indebited to my advisors, Nicholas Feamster and Mostafa Ammar for completion of this thesis. They have guided me by bringing clarity to my ideas and walking me around when I ran into walls. They inspire me to be their likeness in smartness and dedication.

I am grateful to my thesis reading committee members, Dr. Alexandar Gray, Dr. Walter Willinger, Dr. Ellen Zegura, and my advisors, who spared time to read my disseration, conducted the thesis defense, and provided feedback that improved many aspects of the dissertation.

I am also thankful to my coauthors, Kaushik Bhandankar, Jake Brutlag, Murtaza Motiwala, Natalia Sutin, Vytautus Valancius, and Amgad Zeitoun. Their contributions made possible the papers that form the core of this thesis. Amgad Zeitoun and Jake Brutlag hosted me as an intern at Google during summers of years 2007, 2008, and 2009, and provided an exposure to large-scale practical problems that resulted in ideas and projects for at least two of the chapters in this thesis.

I am also fortunate to have worked with my brilliant labmates, Bilal Anwer, Amogh Dhamdhere, Ahmed Mansy, Yogesh Mundada, and Anirudh Ramachandaran. Although the papers that we coauthored are not part of this dissertation, there are a number of things that I learned in these projects that reflect in the papers that are part of the thesis.

Finally, I'd like to thank Andre Broido at Google for serving as my sounding board and mentor during the time that I spent at Google, and also meticulously reviewing all the three papers that form the core of this thesis dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The performance of Internet-based services depends on many server-side, client-side, and network related factors. Often, the interaction among the factors or their effect on service performance is not known or well-understood. The complexity of these services makes it difficult to develop analytical models. Lack of models impedes network management tasks, such as predicting performance while planning for changes to service infrastructure, or diagnosing causes of poor performance.

We posit that we can use statistical causal methods to model performance for Internet-based services and facilitate performance related network management tasks. Internet-based services are well-suited for statistical learning because the inherent variability in many factors that affect performance allows us to collect comprehensive datasets that cover service performance under a wide variety of conditions. These conditional distributions represent the functions that govern service performance and dependencies that are inherent in the service infrastructure. These functions and dependencies are accurate and can be used in lieu of analytical models to reason about system performance, such as predicting performance of a service when changing some factors, finding causes of poor performance, or isolating contribution of individual factors in observed performance.

We present three systems, What-if Scenario Evaluator (WISE), How to Improve Performance (HIP), and Network Access Neutrality Observatory (NANO), that use statistical causal methods to facilitate network management tasks. WISE predicts performance for what-if configurations and deployment questions for content distribution networks. For this, WISE learns the causal dependency structure among the latency-causing factors, and when one or more factors is changed, WISE estimates effect on other factors using the dependency structure. HIP extends WISE and uses the causal dependency structure to invert the performance function, find causes of poor performance, and help answers questions about how to improve performance or achieve performance goals. NANO uses causal inference to quantify the impact of discrimination policies of ISPs on service performance. NANO is the only tool to date for detecting destination-based discrimination techniques that ISPs may use.

We have evaluated these tools by application to large-scale Internet-based services and by experiments on wide-area Internet. WISE is actively used at Google for predicting network-level and browser-level response time for Web search for new datacenter deployments. We have used HIP to find causes of high-latency Web search transactions in Google, and identified many cases where high-latency transactions can be significantly mitigated with simple infrastructure changes. We have evaluated NANO using experiments on wide-area Internet and also made the tool publicly available to recruit users and deploy NANO at a global scale.

CHAPTER I

INTRODUCTION

Performance of Internet-based systems and services depends on many components and variables. For example, the time it takes for a user to make a request to a Web-based service and the response loading and rendering in user's browser application depends on factors ranging from relative geographical location of the user and the server, the properties the content distribution network, the front-end proxies, the back-end servers, the properties of the content requested by the user, the time-of-the-day and load on the servers or network, the network bandwidth and round-trip time, packet loss rate, the capabilities of user's host machine and applications that the user used to access the service. In the case of a composite service, the overall service performance depends may additionally depend on performance of many networked services, each with its own dependencies. Many of these factors may interact with each other in subtle ways that are often not well understood, and are difficult to model. This makes Internet-based services complex and difficult to manage.

Many network management tasks require reasoning about performance of services hosted on the network, such as predicting performance before investing in infrastructure improvements, or investigating causes of high-latency service responses when trying to achieve service performance goals. Operators of a content distribution network for Internet-based services may wish to evaluate *what-if* scenarios, such as, what would be the impact on the service performance if they deploy a new datacenter, or a front-end proxy server, or change the mapping of a subset of serving datacenter for a subset of users. The network operators may wonder what is unique about high-latency responses or generally poorer service performance in a particular region.

Operators would like to know *how-to* modify an existing deployment to achieve certain service performance goals. Similarly, the service providers and users of services may wish to quantify the isolated effect various risk factors for service performance; for example, the users may wish to determine whether their Internet service provider (ISP) is discriminating against a service, and may wish to quantify the impact of such discrimination on the service performance.

Answering these network management questions requires understanding the dependencies among system components and factors that determine the performance of the service. To evaluate the *what-if* scenarios, dependencies among the factor determines whether and how changing one factor would impact other factors and eventually the service performance. Similarly, answering *how-to* questions and finding causes of poor performance requires knowing what factors affect the performance. Unfortunately, because Internet-based services are complex, factors that affect service performance, or the nature of dependencies among these factors is not known or well understood.

It is difficult to develop analytical models for performance of networked services because networked systems are complex, making models that are accurate yet simple enough to understand a nearly intractable task. Developing sufficiently accurate analytical models for even simple networked systems can take a long time; for example TCP throughput prediction presents many challenges even after many years of research [44, 11, 25, 4, 39]. Even if one invests time in developing analytical models for Internet-based services, they may become obsolete very quickly as the services evolve and new versions are rolled out.

This dissertation presents tools that address the problem of modeling and reasoning about performance of ever changing Web based services. Instead of using analytical models, we use statistical and machine learning based models to understand the functional and structural dependencies among factors affecting performance for

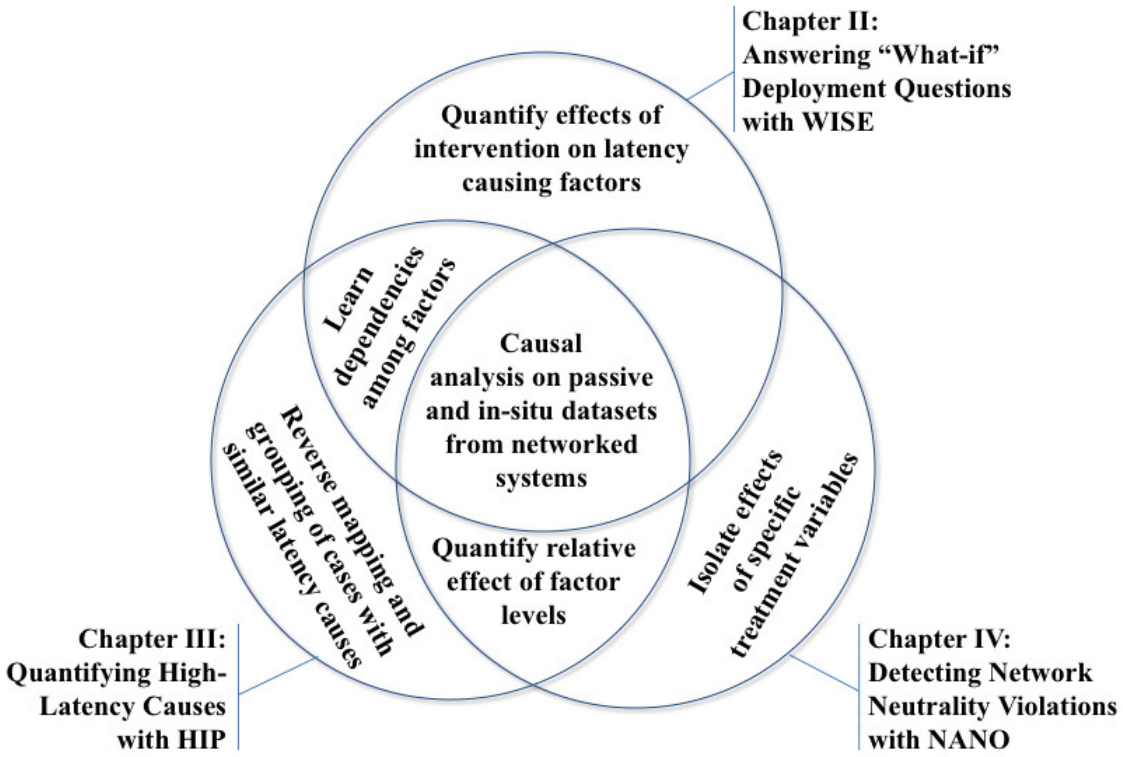

Chapter II: Answering "What-if" Deployment Questions with WISE
Quantify effects of intervention on latency causing factors
Learn dependencies among factors
Causal analysis on passive and in-situ datasets from networked systems
Reverse mapping and grouping of cases with similar latency causes
Quantify relative effect of factor levels
Isolate effects of specific treatment variables
Chapter III: Quantifying High-Latency Causes with HIP
Chapter IV: Detecting Network Neutrality Violations with NANO

Figure 1: Main contributions and underlying techniques in this dissertation.

Internet-based services. The following summarizes the central thesis and key contributions in this dissertation.

Thesis. *We posit that we can use statistical causal methods to facilitate complex network performance management tasks for Internet-based systems. These systems are well-suited for statistical learning because the inherent variability in factors that affect performance, allows us to collect comprehensive datasets that cover system performance under a wide variety of conditions. These conditional distributions represent the functions that govern system performance and dependencies that are inherent in the system. These functions and dependencies are accurate and can be used in lieu of analytical models to reason about system performance and facilitate network management tasks, such as predicting performance of a service when changing some factors, finding causes of poor performance, or isolating contribution of factors in* *performance.*

Contributions. To support this thesis, we solve a number of complex network management problems using causal analysis of passively collected data from networked systems. Figure 1 shows the main contributions in this dissertation and the causal analysis techniques that they use. We present three systems: What-if Scenario Evaluator (WISE), How-to for Improving Performance (HIP) and Network Access Neutrality Observatory (NANO). WISE uses causal analysis to answer *what-if* questions and predict performance for hypothetical configuration and deployment questions for content distribution networks (CDN). HIP extends the causal analysis in WISE to help CDN operators answer *how-to* questions. HIP finds causes of high latency using the structure of dependencies among causal variable that WISE discovers. NANO uses causal inference to quantify the impact of ISP discrimination policies on service performance. For this, NANO adjusts for confounding variables that may confuse the causal inference.

Figure 1 highlights the thematic topics that WISE, HIP, and NANO cover. They extensively leverage existing literature in causal analysis and develop techniques to solve problems in domain of networked systems. The central theme in all three tools is causal analysis on passive and in-situ datasets collected from existing networked systems. WISE and HIP both rely on structure of causal dependencies among variables in the system. WISE uses this causal structure to guide statistical intervention evaluation, which quantifies the effect of changes in one or more of the variables for answering *what-if* questions. HIP uses the causal structure to find a reverse mapping from observed response time to potential causes. HIP also finds groups of Web requests that have similar underlying causes for poor performance and isolates those causes. HIP and NANO both quantify the effect of causes by comparing values under multiple values (levels) of a factor. HIP uses this comparison to identify the causes that are responsible for poor performance. NANO uses this comparison to determine whether an ISP is discriminating against a service resulting in poorer performance relative to other ISPs. NANO also quantifies the effect that the ISP has on service performance.

We have evaluated these tools by application to large-scale Internet-based services and through experiments on wide-area Internet. WISE is a active production level tool at Google for evaluating *what-if* scenarios for new deployments. We have used HIP to identify causes of poor response time for Web Search service at Google and obtained encouraging results. NANO is the only tool to date for detecting destination based discriminations that ISPs may use. We have evaluated NANO using experiments on wide-area Internet and also made the tool publicly available to recruit users and deploy NANO at a global scale. Following presents a brief overview of these tools.

Evaluating What-if Deployment and Configuration Questions and Applications with WISE. What-if Scenario Evaluator (WISE), automatically learns the dependencies among the various factors that determine the service response time using the datasets obtained from existing deployments. WISE shields the network designers and operators from the intricate details of these dependencies by presenting the network designers with an easy to use interface in form of WISE Specification Language (WSL). Using WSL the operators can specify a typical *what-if* scenario in two to three lines of code. WISE then evaluates the scenario, automatically taking care of potentially several layers of the functional dependencies among the factors, ultimately producing the distribution of the response time for the desired scenario. We have used WISE to predict network-level and browser-level response time for Google's Web search service. We find that WISE produces highly accurate predictions. WISE is an active production-level tool used for predicting performance for new deployments.

Quantifying High-Latency Causes with HIP. How-to for Improving Performance (HIP) is an extension to WISE. HIP answers the *how-to* questions about performance of networked services. HIP uses the causal dependency structure that WISE produces and clusters the high-latency responses based on the similarity of causal variables. HIP then identifies the responsible causal variable for each cluster that is produced. HIP separates systemic causes from transient ones and produces hypothetical values for factors that can improve the performance for high-latency requests. We have used HIP to identify causes for high-latency for Google's Web search service. We find that HIP can find the most important causes of high-latency for each region. HIP can also determine the subset of users or servers that are affected by systemic causes. Using a reference distribution as input, HIP can suggest values for responsible causal variables that can potentially alleviate high-latency responses.

Detecting Network Neutrality Violations using NANO. Network Access Neutrality Observatory (NANO) detects and quantifies the effect of ISP discrimination on service performance. NANO uses causal inference on passive monitoring data about performance of network based services from participating users for this inference. NANO adjusts for confounding variables, such as user location, application, operating system, time-of-the-day, and estimates the difference in performance of services among ISPs. NANO can also provide clues about the criteria that the ISP might be using for discrimination. NANO is only network neutrality detection tool to date that can detect destination-based discrimination. Also because NANO uses passively collected performance data for inference, it is robust to *whitelisting* of probing traffic by ISP. Using experiments on PlanetLab and Emulab we demonstrate that NANO can detect neutrality violations and quantify the effect of these policies on performance. We have also made NANO-client implementation freely available to recruit real users and deploy NANO on global scale.

Rest of this dissertation is organized as following. Chapter 2 presents WISE, Chapter 3 presents HIP and Chapter 4 presents NANO. We discuss related work for our thesis and individual tools in Chapter 5. We conclude the dissertation in Chapter 6.

CHAPTER II

ANSWERING "WHAT-IF" DEPLOYMENT AND CONFIGURATION QUESTIONS WITH WISE

Designers of content distribution networks often need to determine how changes to infrastructure deployment and configuration affect service response times when they deploy a new data center, change ISP peering, or change the mapping of clients to servers. Today, the designers use coarse, back-of-the-envelope calculations, or costly field deployments; they need better ways to evaluate the effects of such hypothetical "what-if" questions before the actual deployments. This chapter presents *What-If Scenario Evaluator (WISE)*, a tool that predicts the effects of possible configuration and deployment changes in content distribution networks. WISE makes three contributions: (1) an algorithm that uses traces from existing deployments to learn causality among factors that affect service response-time distributions; (2) an algorithm that uses the learned causal structure to estimate a dataset that is representative of the hypothetical scenario that a designer may wish to evaluate, and uses these datasets to predict hypothetical response-time distributions; (3) a scenario specification language that allows a network designer to easily express hypothetical deployment scenarios without being cognizant of the dependencies between variables that affect service response times. Our evaluation, both in a controlled setting and in a real-world field deployment on a large, global CDN, shows that WISE can quickly and accurately predict service response-time distributions for many practical *what-if* scenarios.

2.1 Introduction

Content distribution networks (CDNs) for Web-based services comprise hundreds to thousands of distributed servers and data centers [2, 7, 21]. Operators of these networks continually strive to improve the response times for their services. To perform this task, they must be able to predict how service response-time distribution changes in various hypothetical *what-if* scenarios, such as changes to network conditions and deployments of new infrastructure. In many cases, they must also be able to reason about the *detailed* effects of these changes (e.g., what fraction of the users will see at least a 10% improvement in performance because of this change?), as opposed to just coarse-grained point estimates or averages.

Many factors affect a CDN's service response time on both short and long time scales. On short time scales, response time can be affected by routing instability or changes in server load. Occasionally, the network operators may "drain" a data center for maintenance and divert the client requests to an alternative location. Over longer time scales, service providers may upgrade their existing facilities, move services to different facilities or deploy new data centers to address demands and application requirements, or change peering and customer relationships with neighboring ISPs. These instances require significant planning and investment; some of these decisions are hard to implement and even more difficult to reverse.

Unfortunately, reasoning about the effects of any of these changes is extremely challenging in practice. Content distribution networks are complex systems, and the response time perceived by a user can be affected by a variety of inter-dependent and correlated factors. Such factors are difficult to accurately model or reason about and back-of-the-envelope calculations are not precise.

This chapter presents the design, implementation, and evaluation of *What-If Scenario Evaluator* (WISE), a tool that estimates the effects of possible changes to network configuration and deployment scenarios on service response time. WISE uses statistical learning techniques to provide a way of interpreting *what-if* questions as statistical interventions. WISE uses packet traces from Web transactions to prepare a graphical causal and functional model for the factors that affect the service response time. Network designers can then use WISE to specify a *what-if* scenario that changes one or more of the factors. Using the causal and functional model for the CDN, WISE determines new statistical distributions for factors (including the response time) that are directly or transitively affected by the changes specified by the designer. Because WISE is aware of the causal functional model of the CDN, the new distributions of the variables represent the *what-if* scenario and are consistent with the operational CDN.

Although function estimation using passive datasets is a common application in machine learning, using these techniques is not straightforward because they can only accurately predict the response-time distribution for a *what-if* scenario if the estimated function receives accurate *joint distribution of all the factors that affect response time* as input. Providing this joint distribution of factors as input presents several challenges:

First, **WISE must allow the network designers to easily specify *what-if* scenarios**. To use a typical off-the-shelf function estimator for evaluating a *what-if* scenario, a designer would have to provide an accurate joint distribution of all the factors that affect the response time, such that this distribution is representative of the scenario as well as consistent with dependencies that exist in the operational CDN. For example, packet loss and retransmit rates depend on network round-trip time and bandwidth. For consistency, if a *what-if* scenario changes network round-trip time or bandwidth, the packet loss or retransmits should also change appropriately during evaluation of the scenario. Achieving this goal is hard in practice, because the designer to specify a *what-if* scenario as a change to one or a few factors relative to an existing "baseline" CDN deployment but be unaware that *how* the change might also affect other related factors. WISE's interface shields the designers from these complex details. WISE provides a *scenario specification language* that allows network designers to succinctly specify hypothetical scenarios for arbitrary subsets of existing networks and to specify *what-if* values for different features. WISE's specification language is simple: evaluating a hypothetical deployment of a new proxy server for a subset of users can be specified in only 2 to 3 lines of code.

Second, because the designer can specify a *what-if* scenario without being aware of dependencies among the factors affecting response time, **WISE must automatically produce an accurate joint distribution of all factors affecting the response time** that is both *representative* of the *what-if* scenario the designer specifies and *consistent* with the causal and functional dependencies in the CDN. To enforce this consistency, WISE uses a causal dependency discovery algorithm to discover the dependencies among variables and a statistical intervention evaluation technique to transform the observed dataset to a representative and consistent dataset representing the joint distribution. Once this dataset is ready, WISE estimates the response-time distribution for the *what-if* scenario.

We have used WISE to predict service response times in both controlled settings on the Emulab testbed and for Google's global CDN for its Web search service. Our evaluation shows that WISE's predictions of response-time distribution are very accurate, yielding less than 5% error in estimating the mean response-time. Median error for estimating response-time for individual requests is between 8% and 11% for cross-validation with existing deployments and only 9% maximum cumulative distribution difference compared to ground-truth response time distribution for *what-if* scenarios on a real deployment as well as controlled experiments on Emulab.

Finally, **WISE must be fast**, so that it can be used for short-term and frequently arising questions. To achieve this, WISE uses approximations that enable fast computations without sacrificing the accuracy significantly. Further, we have tailored WISE for parallel computation and implemented it using the Map-Reduce [16] framework, which allows us to process large datasets comprising hundreds of millions of records quickly and produce accurate predictions for response-time distributions.

Since the original publication [54], WISE has been used at Google for evaluating real *what-if* scenarios. Real world experience led to improvements to original work. These include enhancements to the prediction techniques and using WISE to for additional applications. With regards to prediction, we found that our initial function estimation based on Kernel Regression (KR) is accurate, but sometimes it is difficult to have training data that covers prediction for rare combinations of variables in the system. To resolve this problem, we have extended WISE to use an approximated nearest-neighbor based approach (Section 2.5.5). This approach requires less data, is comparable in accuracy to KR for estimating distribution of variables, and is also faster in terms of computation time. We also extend WISE to predict *browser-level response time* (*brt*), in addition to the network-level response time (*nrt*) for Google's CDN. Browser-level response time is a better metric to estimate user experience, but *brt* is also more volatile than *nrt* because *brt* depends on additional factors, such as type of browser, DNS latency, cached content, and computational capability and load on client. Some of these factors are not visible to the CDN operator. We describe our extensions to WISE for estimating *brt* and show that WISE accurately predicts *brt*. This chapter also presents results of applying WISE to predict network-level response time for ten additional countries and for predicting browser-level response time for seven different countries.

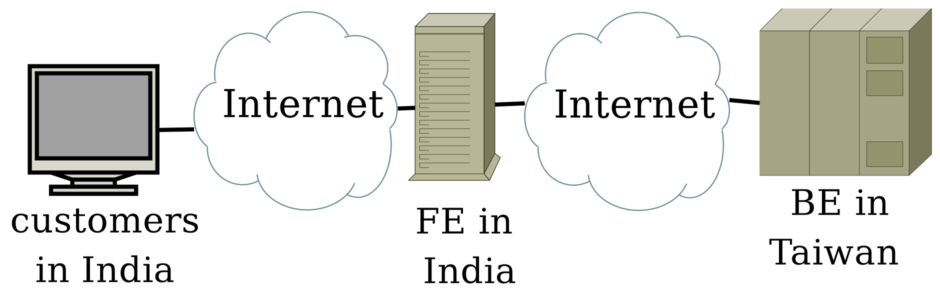
The chapter proceeds as follows. Section 2.2 describes the problem scope and motivation. Section 2.3 makes the case for using statistical learning for *what-if* scenario evaluation. Section 2.4 provides an overview of WISE, and Section 2.5 describes WISE's algorithms in detail. We discuss the implementation in Section 2.7. In Section 2.8, we evaluate WISE for response-time estimation for existing deployments as well as for a *what-if* scenario based on a real operational event. In particular, we demonstrate that WISE can accurately predict response time for network level transfer of data, as well as the latency for loading and rendering of a Web page when a user makes a query to Google's Web search service. In Section 2.9, we evaluate WISE for *what-if* scenarios for a small-scale but controlled network environment built on the Emulab testbed. In Section 2.10, we discuss various properties of the WISE system and how it relates to other areas in networking.

2.2 Problem Context and Scope

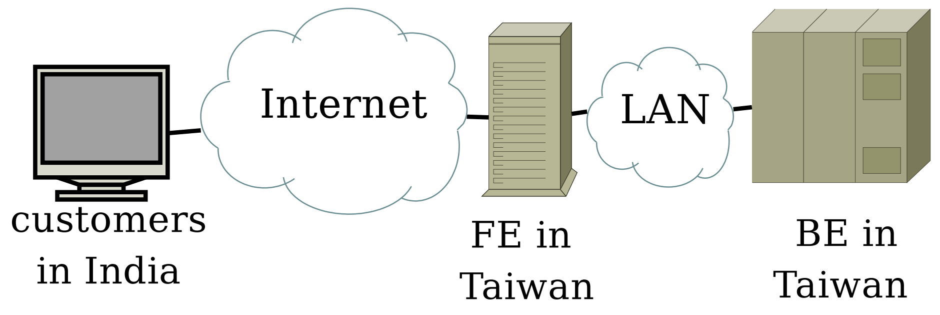
This section describes common *what-if* questions that the network designers pose when evaluating potential configuration or deployment changes to an existing content distribution network deployment.

Content Distribution Networks. Most CDNs conform to a two-tier architecture. The first tier comprises a set of globally distributed front-end (FE) servers that, depending on the specific implementation, provide caching, content assembly, pipelining, request redirection, and proxy functions. The second tier comprises back-end (BE) servers that implement the application logic, and which might also be replicated and globally distributed. The FE and BE servers may belong to a single administrative entity (as is the case with Google [7]) or to different administrative entities, as with commercial content distribution networking service providers, such as Akamai [2]. The network path between the FE and BE servers may be over a public network or a private network, or a LAN when the two are colocated. CDNs typically use DNS redirection or URL-rewriting [6] to direct the users to the appropriate FE and BE servers; this redirection may be based on the user's proximity, geography, availability, and relative server load.

An Example "What-if" Scenario. The network designers may want to ask a variety of *what-if* questions about the CDN configuration. For example, the network



(a) Before the Maintenance



(b) During the Maintenance

Figure 2: A *what-if* scenario for network configuration of customers in India.

designers may want to determine the effects of deploying new FE or BE servers, changing the serving FE or BE servers for a subset of users, changing the size of typical responses, increasing capacity, or changing network connectivity, on the service response time. We now present an actual *what-if* scenario from Google's CDN for the Web-search service.

Figure 2 shows an example of a change in network deployment that could affect server response time. Google has an FE data center in India that serves users in India and surrounding regions. This FE data center uses BE servers located elsewhere in the world, including the ones located in Taiwan. On July 16, 2007, the FE data center in India was temporarily "drained" for maintenance, and the traffic was diverted to a FE data center that is colocated with a BE in Taiwan, resulting in a change in latency for the users in India. This change in the network configuration can be described as a what-if scenario in terms of change of the assigned FE, or more explicitly as changes in delays between FE and clients that occur due to the new configuration. WISE aims to predict the response-time distribution for reconfigurations before they are deployed in practice.

2.3 A Case for Machine Learning

In this section, we present two aspects of *what-if* scenario evaluation that make the problem well-suited for machine learning: (1) an underlying model that is difficult to derive from first principles but provides a wealth of data; (2) a need to predict outcomes based on data that may not directly represent the desired *what-if* scenario.

The system is complex, but observable variables are driven by fundamental properties of the system. Unfortunately, in large complex distributed systems such as CDNs, the parameters that govern the system performance, the relationships between these variables, and the functions that govern the response-time distribution of the system are often complex and characterized by randomness and variability that are difficult to model as simple, readily evaluatable formulas.

Fortunately, the underlying fundamental properties and dependencies that determine a CDN's response time can be observed as correlations and conditional probability distributions of the variables that define the system, including the service response time. By observing these conditional distributions (e.g., response times observed under various conditions), machine learning algorithms can infer the underlying function that affects the response time. Naturally occurring variability in the system parameters allows observing distributions of variables under a variety of conditions. Because most production CDNs collect comprehensive datasets for their services as part of everyday operational and monitoring needs, the requisite datasets are typically readily available.

Obtaining datasets that directly represent the *what-if* scenario is challenging. Once the response-time function is learned, evaluating a *what-if* scenario requires providing this function with input data that is representative of the joint distribution of all the factors affecting the response time under the *what-if* scenario. Unfortunately, joint distribution in dataset collected from an existing network deployment only represents the current setup, and the system complexities make it difficult for a designer to manually "transform" the data to represent the new scenario.

Fortunately, depending on the extent of the cp;;ected data and the nature of *what-if* scenario, machine learning algorithms can reveal the dependencies among variables and use the dependency structure to intelligently *re-weigh and re-sample* the different parts of the existing dataset to perform this transformation. In particular, if the *what-if* scenario is expressed in terms of the changes to values of the variables that are observed in the dataset and the changed values or similar values of these variables are observed in the dataset even with small densities in the original dataset, then we can transform the original dataset to one that is representative of the *what-if* scenario as well as the underlying principles of the system, while requiring minimal input from the network designer.

2.4 WISE: High-Level Design

WISE entails four steps: (1) identifying features in the dataset that affect response time; (2) constraining the inputs to "valid" scenarios based on existing dependencies; (3) specifying the *what-if* scenario; (4) estimating the response-time function and distribution. Each of these tasks raises a number of challenges, some of which are general problems with applying statistical learning in practice, and others are specific to *what-if* scenario evaluation. This section provides an overview and necessary background for these steps. Section 2.5 discuss the mechanisms in more detail.

1. Identifying Relevant Features. The main input to WISE is a comprehensive dataset that covers many combinations of variables. Most CDNs have existing network monitoring infrastructure that can typically provide such a dataset. This dataset, however, may contain variables that are not relevant to the response-time function. WISE extracts the set of relevant variables from the dataset and discards the rest of the variables. WISE can also identify whether there are missing or latent variables that may hamper scenario evaluation (Sections 2.5.1 and 2.5.2 provide more details).

The nature of *what-if* scenarios that WISE can evaluate is limited by the input dataset—careful choice of variables that the monitoring infrastructure collects from a CDN can therefore enhance the utility of the dataset for evaluating *what-if* scenarios, choosing such variables is outside the scope of WISE system.

2. Preparing Dataset to Represent the What-if Scenario. Evaluating a *what-if* scenario requires values for input variables that "make sense." Specifically, an accurate prediction of the response-time distribution for a *what-if* scenario requires a joint distribution of the input variables that is representative of the scenario and is also consistent with the dependencies that are inherent to the system itself. For instance, the distribution of the number of packets that are transmitted in the duration of a service session depends on the distribution of the size of content that the server returns in reply to a request; if the distribution of content size changes, then the distribution for the number of packets that are transmitted must also change in a way that is inherent to the system, e.g., the path-MTU might determine the number of packets. Further, the change might *cascade* to other variables that in turn depend on the number of packets. To enforce such consistency WISE learns the dependency structure among the variables and represents these relationships as a *Causal Bayesian Network (CBN)* [45]. We provide a brief background of CBN in this Section and explain the algorithm for learning the CBN in Section 2.5.2.

A CBN represents the variables in the dataset as a Directed Acyclic Graph (DAG). The nodes represent the variables and the edges indicate whether there are dependencies among the variables. A variable has a "causal" relationship with another

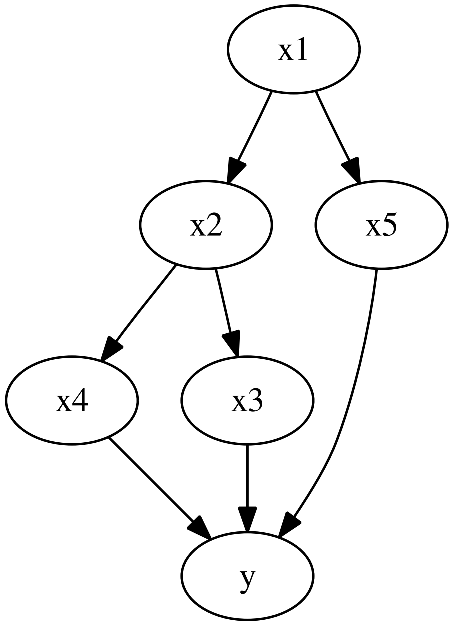


Figure 3: Example of a causal DAG.

variable, if a change in the value of the first variable *causes* a change in the values of the later. When conditioned on its parent variables, a variable $\mathbf{x_i}$ in a CBN is independent of all other variables in the DAG except its descendents. An optimal DAG for a dataset is one where we find the minimal parents for each node that satisfy the above property.

As an example of how the causal structure may facilitate scenario specification and evaluation, consider a dataset with five input variables ($x_1 \ldots x_5$), and target variable y.

Suppose that we discover a dependency structure among them as shown in Figure 3. If WISE is presented with a *what-if* scenario that requires changes in the value of variable x_2, then the distributions for variables x_1 and x_5 remains unchanged in the input distribution, and WISE needs to update only the distribution of the descendants of x_2 to maintain consistency. WISE constrains the input distribution by intelligently re-sampling and re-weighing the dataset using the causal structure as a guideline (see Section 2.5.4).

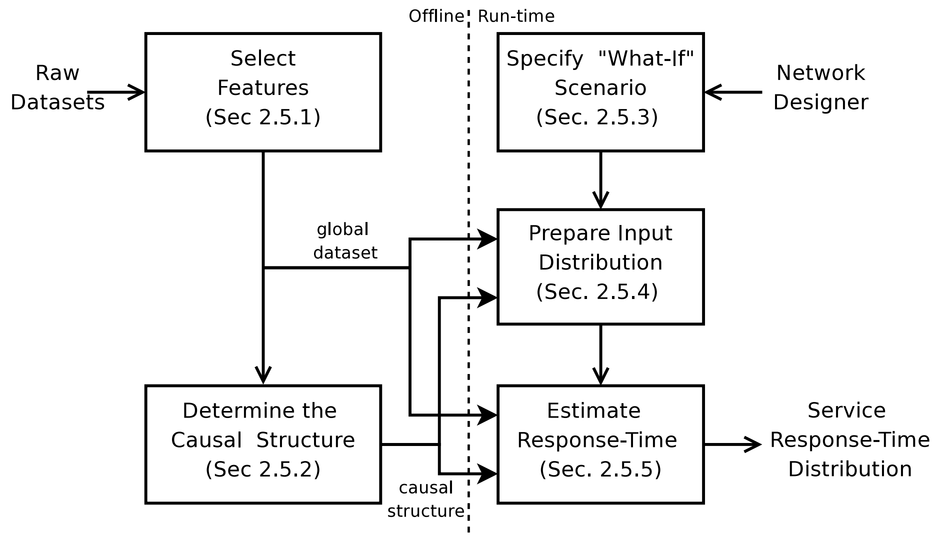

Offline
Run-time
Raw Datasets
Select Features (Sec 2.5.1)
Specify "What-If" Scenario (Sec. 2.5.3)
Network Designer
global dataset
Prepare Input Distribution (Sec. 2.5.4)
Determine the Causal Structure (Sec 2.5.2)
Estimate Response-Time (Sec. 2.5.5)
Service Response-Time Distribution
causal structure

Figure 4: WISE approach.

3. Facilitating Scenario Specification. WISE presents the network designers with an easy-to-use interface in the form of a scenario specification language called WISE-Scenario Language (WSL). The designers can typically specify the baseline setup as well as the hypothetical values for the scenario in 3-4 lines of WSL.

WSL allows the designers to evaluate a scenario for an arbitrary subset of customers. WSL also provides a useful set of built-in operators that facilitate scenario specification as relative changes to the existing values of variables or as new values from scratch. With WSL, the designers are completely shielded from the complexity of dependencies among the variables, because WISE automatically updates the dependent variables. We detail WSL and the process of scenario specification and evaluation in Sections 2.5.3 and 2.5.4.

4. Scalable Estimation of New Distributions for Variables. Datasets for typical CDN deployments and *what-if* scenarios span a large multi-dimensional space. While non-parametric function estimation is a standard application in the machine learning literature, the computational requirements for accurately estimating a function spanning such a large space can be astronomical. To address this, WISE estimates the function in a piece-wise manner, and also structures the processing so that it is amenable to parallel processing. WISE also uses the dependency structure to reduce the number of variables that form the input to the regression function. Sections 2.5.5 and 2.6.1 provide more detail.

2.5 WISE: System

This section describes the algorithmic and design details for each of the steps in WISE framework. Section 2.5.1 presents feature selection, Section 2.5.2 presents the WISE Causal Discovery (WCD) Algorithm and Section 2.5.3 describes the WISE Scenario Specification Language (WSL). Sections 2.5.4 and 2.5.5 present how WISE evaluates a scenario to predict distributions for response time and other intermediate variables.

2.5.1 Feature Selection

Traditional machine-learning applications use various model selection criteria, such as Akaike Information Criterion (AIC), Mallow's C_p Test, or k-fold cross-validation [55], to determine the appropriate subset of covariates for a learning problem. WISE foregoes the traditional model selection techniques in favor of simple pair-wise independence testing, because at times the conventional techniques can ignore variables that might allow the designer to more easily interpret the results.

WISE uses simple pair-wise independence tests on all the variables in the dataset with the response-time variable and discards all variables that it deems independent of the response-time variable. For each categorical variable (variables that do not have numeric meanings) in the dataset, such as country of origin of a request or AS number, WISE obtains the conditional distributions of response time for each categorical value, and discards the variable if all the conditional distributions of response time are statistically similar. To test similarity, we used Two-sample Kolmogorov-Smirnov (KS) goodness-of-fit test with a significance level of 10%.

For real-valued variables, WISE first tests for correlation with the response-time variable, and retains a variable if the correlation coefficient is greater than 10%. Unfortunately, for continuous variables, lack of correlation does not imply independence, so we cannot outright discard a variable if we observe small correlation. A typical example of such a variable in a dataset is the timestamp of the Web transaction, where the correlation may cancel out over a diurnal cycle. For such cases, we divide the range of the variable in question into small buckets and treat each bucket as a category. We then apply the same techniques as we do for the categorical variables to determine whether the variable is independent. There is still a possibility that we may discard a variable that is relevant, but this outcome is less likely if sufficiently small buckets are used. The bucket size depends on the variable in question; for instance, we use one-hour buckets for the timestamp variable in the datasets.

2.5.2 Learning the Causal Structure

To learn the causal structure, WISE first learns the undirected graph and then uses a set of rules to orient the edges.

Learning the Undirected Graph. Recall that in a Causal Bayesian Network (CBN), a variable, when conditioned on its parents, is independent of all other variables, except its descendants. Further an optimal CBN requires finding the smallest possible set of parents for each node that satisfy this condition. Thus by definition, variables a and b in the CBN have an edge between them, if and only if, there is a subset of *separating variables*, S_{ab}, such that a is independent of b given S_{ab}. This, in the general case, requires searching all the possible $O(2^n)$ combinations of the n variables in the dataset

WISE-Causal Discovery Algorithm (WCD) (Figure 5) uses a heuristic to guide the

```
1: WCD ($\mathbf{V}, \mathbf{W_0}, \mathbf{\Delta}$)
     /*Notation
      $\mathbf{V}$: set of all variables
      $\mathbf{W_0}$: set of no-cause variables
      $\mathbf{\Delta}$: maximum allowable cardinality for separators
      $a \perp b$: Variable $a$ is independent of variable $b$ */
2:   Make a complete Graph on $\mathbf{V}$
3:   Remove all edges $(a, b)$ if $a \perp b$
4:   $\mathbf{W} = \mathbf{W_0}$
5:   for c = 1 to $\mathbf{\Delta}$ /*prune in the order of increasing cardinality*/
6:     LocalPrune ($c$)

1: LocalPrune ($c$)
     /*Try to separate neighbors of frontier variables $\mathbf{W}$*/
2:   $\forall w \in \mathbf{W}$
3:     $\forall z \in N(w)$ /*neighbors of $w$*/
4:       if $\exists \mathbf{x} \subseteq N(z) \backslash w : |\mathbf{x}| \leq c, z \perp w|\mathbf{x}$
5:         then /*found separator node(s)*/
             $S_{wz} = \mathbf{x}$ /*assign the separating nodes*/
6:           Remove the edge $(w, z)$
7:           Remove edges $(w', z)$, for all the nodes $w' \in W$ that are also on path from
   $w$ to nodes in $W_0$
           /*Update the new frontier variables*/
8:           $\mathbf{W} = \mathbf{W} \cup \mathbf{x}$
```

Figure 5: WISE Causal Discovery (WCD) algorithm.

search of separating variables when we have prior knowledge of a subset of variables that are "not caused" by any other variables in the dataset, or that are determined by factors outside our system model (we refer to these variables as the *no-cause* variables). Further, WCD does not perform exhaustive search for separating variables, thus forgoing optimality for lower complexity.

WCD starts with a fully connected undirected graph on the variables and removes the edges among variables that are independent. WCD then progressively finds separating nodes between a restricted set of variables (that we call *frontier* variables), and the rest of the variables in the dataset, in the order of increasing cardinality of allowable separating variables. Initially the frontier variables comprise only the no-cause variables. As WCD discovers separating variables, it adds them to the set of frontier variables.

The algorithm terminates when it has explored separation sets up to the maximum allowed cardinality $\mathbf{\Delta} \leq n$, resulting in a worse case complexity of $O(2^{\Delta})$. This termination condition means that certain variables that are separable are not separated: this does not result in false dependencies but potentially transitive dependencies may be considered direct dependencies. This sub-optimality does not affect the accuracy of the scenario datasets that WISE prepares, but it reduces the efficiency because it leaves the graph to be denser and the nodes having larger in-degree. Because the WISE needs values of all the parent variables to determine the value of a child variable, determining value of high in-degree child variables requires knowing value of many variables. This, as we will discuss in section 2.6.1, increases the data requirement for accurate prediction. Fortunately, because WCD is part of the off-line steps in WISE, we can afford a large maximum cardinality (Δ) search for the separating nodes; this will improve the efficiency for the online stage of scenario evaluation.

In the cases where the set of no-cause variables is unknown, WISE relies on the PC-algorithm [52], which also performs search for separating nodes in the order of increasing cardinality among all pair of variables, but not using the frontier variables.

Orienting the Edges. WISE orients the edges and attempts to detect latent variables using the following simple rules, well known in the literature. We reproduce these rules, slightly adapted to accommodate heuristics in WCD, and refer the reader to [45] for further details.

1. Add outgoing edges from the no-cause variables.

2. If node c has nonadjacent neighbors a and b, and $c \in S_{ab}$, then orient edges $a \rightarrow c \leftarrow b$ (unmarked edges).

3. For all nonadjacent nodes, a, b, with a common neighbor c, if there is an edge from a to c, but not from b to c, then add a marked edge $c \overset{*}{\rightarrow} b$.

4. If a and b are adjacent and there is directed path of only marked edges from a to b, then add $a \rightarrow b$

In the resulting graph, any unmarked, bi-directed, or undirected edges signify possible latent variables and *ambiguity in causal structure*. In particular, $a \rightarrow b$ means either a really causes b or there is a common latent cause L causing both a and b. $a \leftrightarrow b$, signifies a definite common latent cause, and undirected edge between a and b implies either a causes b, b causes a, or a common latent cause in the underlying model.

We address issues related to missing variables or causal relationship that may arise due to limitations of dataset in section 2.6.2. Section 2.5.4 discusses how WISE deals with ambiguities in causal structure.

2.5.3 Specifying the "What-If" Scenarios

Figure 6 shows the grammar for WISE-Specification Language (WSL). A scenario specification with WSL comprises a *use*-statement, followed by optional scenario *update*-statements.

The *use*-statement specifies a condition that describes the subset of present network for which the designer is interested in evaluating the scenario. This statement provides a powerful interface to the designer for choosing the baseline scenario: depending on the features available in the dataset, the designer can specify a subset of network based on location of clients (such as country, network address, or AS number), the location of servers, properties of service sessions, or a combination of these attributes.

The *update*-statements allow the designer to specify *what-if* values for various variables for the service session properties. Each scenario statement begins with either the INTERVENE, or the ASSUME keyword and allows conditional modification of exactly one variable in the dataset.

```
scenario = use_stmt {update_stmt};
use_stmt = "USE" ("*" | condition_stmt)<EOL>;
update_stmt = ("ASSUME"|"INTERVENE") (set_directive | setdist_directive)
    [condition_stmt]<EOL>;
set_directive = "SET" ["RADIAL"* | "FIXED"] var set_op value;
setdist_directive = "SETDIST" feature dist_name([param]) |
    "FILE" filename);
condition_clause = "WHERE" condition;
condition = simple_cond | compound_cond;
simple_cond = compare_clause | (simple_cond);
compound_cond = (simple_cond ("AND"|"OR") (simple_cond|compound_cond));
compare_clause = (var rel_op value) |  membership_test;
membership_test = feature "IN" (value {,value});
set_op = "+=" | "-=" | "*=" | "\=" | "=";
rel_op = "<=" | ">=" | "<>" | "==" | "<" | ">";
var = a variable from the dataset;
```

Figure 6: Grammar for WISE Specification Language (WSL).

When the statement begins with the INTERVENE keyword, WISE first updates the value of the variable in question. WISE then uses the causal dependency structure to make the dataset *consistent* with the underlying dependencies. For this WISE uses a process called *Statistical Intervention Effect Evaluation* (Section 2.5.4).

Advanced designers can override the intelligent update behavior by using the ASSUME keyword in the update statement. In this case WISE updates the distribution of the variable specified in the statement but does not attempt to ensure that the distribution of the dependent variables are correspondingly updated. WISE allows this functionality for cases where the designers believe that the scenario that they wish to evaluate involves changes to the underlying invariant laws that govern the system. If a variable's only descendant is the target variable then the two types of update statements behave identically. Examples of scenario specification with WSL will follow in Section 2.8.

2.5.4 Preparing the Input Distribution

This section describes how WISE uses the dataset, the causal structure, and the scenario specification from the designer to prepare a meaningful dataset for the *what-if* scenario.

Filtering Dataset to obtain baseline scenario. WISE filters the global dataset for the entries that match the conditions specified in the *use-statement* of the scenario specification to create the *baseline* dataset.

Evaluating Scenario for Baseline Dataset as Statistical Intervention. WISE executes the update-statements, one statement at a time, to change the baseline dataset. To ensure consistency among variables after every INTERVENE update statement, WISE employs a process called *Statistical Intervention Effect Evaluation*; the process is described below.

Let us denote the action requested on a variable $\mathbf{x_i}$ in the *update*-statement as $\mathbf{set}(\mathbf{x_i})$. We refer to $\mathbf{x_i}$ in $\mathbf{set}(\mathbf{x_i})$ as the *intervened* variable. Let us also denote the set of variables that are children of $\mathbf{x_i}$ in the CBN for the dataset as $\mathcal{C}(\mathbf{x_i})$, and the variables that are parents of $\mathbf{x_i}$ in the CBN as $\mathcal{P}(\mathbf{x_i})$. Lets also denote the joint distribution of parent variables of child variables of variable $\mathbf{x_i}$ that has variable $\mathbf{x_i}$ changed, but other parent variables not changed, as: $\{\mathcal{P}(\mathcal{C}(\mathbf{x_i})), \mathbf{set}(\mathbf{x_i})\}$.

If the causal structure is correct then the new distribution of children of $\mathbf{x_i}$ is given as:
$\mathbf{Pr}\{\mathcal{C}(\mathbf{x_i})|\{\mathcal{P}(\mathcal{C}(\mathbf{x_i})), \mathbf{set}(\mathbf{x_i})\}\}$. The intuition is that because the parent node in a CBN has a causal effect on its descendent nodes, we expect that a change in the value of the parent variable must cause a change in the value of the children. Further, the new distribution of children variables would be one that we would expect to observe under the *changed* values of the parent variable that has been changed, and *existing* values of the parent variable(s) that have not been changed.

Because the causal effect cascades to all the descendants of $\mathbf{x_i}$, WISE repeats this process recursively, considering $\mathcal{C}(\mathbf{x_i})$ as the intervened variables and updating the distributions of $\mathcal{C}(\mathcal{C}(\mathbf{x_i}))$, and so on, until all the descendants of $\mathbf{x_i}$ (except the target variable) are updated. WISE cannot update the distribution of a descendant of $\mathbf{x_i}$ until the distribution of all of its ancestors that are descendant of $\mathbf{x_i}$ has been updated. WISE thus carefully orders the sequence of the updates by traversing the CBN DAG in breadth-first order, beginning at node $\mathbf{x_i}$.

WISE sequentially repeats this process for each statement in the scenario specification. The updated dataset produced after each statement serves as the input dataset for the next statement. Once all the statements are executed, the dataset is the representative joint distribution variables for the entire *what-if* scenario.

Emulating Statistical Intervention as Approximate Nearest-Neighbors Search. To apply statistical intervention, WISE conditions the *global* dataset on the new value of the intervened variable, $\mathbf{set}(\mathbf{x_i})$, and the existing values of the all the other parents of the children of the intervened variable, $\mathcal{P}(\mathcal{C}(\mathbf{x_i}))$, in the baseline dataset to obtain an empirical distribution $\mathbf{Pr}\{\mathcal{C}(\mathbf{x_i})|\{\mathcal{P}(\mathcal{C}(\mathbf{x_i})), \mathbf{set}(\mathbf{x_i})\}\}$. Value of child variables for all the samples in this empirical distribution are equally likely *correct* value for the child variables after intervention. WISE thus chooses one of these samples at random, and assigns to the child variable.

In reality, the number of samples in global dataset whose values match exactly with the condition
$\{\mathcal{P}(\mathcal{C}(\mathbf{x_i})), \mathbf{set}(\mathbf{x_i})\}$ may be very few. As a result, the aforementioned empirical distribution is hard to obtain. To overcome this problem, WISE also considers samples whose value is approximately same as the condition. To find such samples, WISE uses nearest-neighbors search centered around the point specified by the above condition. Because nearest neighbor search is expensive on large datasets, WISE approximates nearest-neighbor search by limiting the scope of search around the specified center.

For this, WISE uses a technique that we call *tiling*. Tiling is described in Section 2.6.1.

Dealing with Ambiguities in Causal Structure. When the causal structure has ambiguities, WISE proceeds as follows.

(a) Bi-directed or Undirected Edges. When the edge between two variables is bi-directed or undirected, WISE maintains the consistency by always updating the distribution of one if the distribution of the other is updated.

(b) Unmarked Directed Edges. Cases of unmarked directed edge, $a \rightarrow b$ are quite common. Recall that this implies that either variable a causes b, or there is a latent variable that causes both a and b. WISE updates the distribution of b assuming that a causes b. The intuition is that if there is indeed a latent variable that causes a and b, and if a scenario specification requires changing variable a, then the scenario specification is implicitly stating a change in that latent variable, and thus changes in a and b are in order. Further, the only way that we can know about the effect of change of the latent variable on b is what we observe in the dataset as effect on b as a result of changes in a.

2.5.5 Estimating Response Time Distribution

Finding the new distribution of response time is just another case of intervention effect evaluation. WISE uses the techniques in Section 2.5.4 considering response time as the affected child variable. Once the response-time distribution is computed, WISE computes the expected response time by simply integrating over the distribution of response time.

Random NN based prediction is very efficient and works great for predicting distributions, but if the designers are interested in predicting response time for individual requests, Random NN can have large error. For these cases, WISE supports using a weighted average of all NN for prediction. In particular, we use a standard Kernel

Regression (KR) method, with a radial basis Kernel function [56] to estimate the response time for each request in the dataset. To address the computational complexity, WISE applies the KR in a piece-wise manner; the details follow in Section 2.6.1.

2.6 Additional Challenges

This section describes additional challenges. Section 2.6.1 presents how WISE addresses scalability challenges that arise due to the size of the required training data. Section 2.6.2 discusses how WISE copes with missing variables or missing causal dependencies that may arise due to insufficient data.

2.6.1 Computational Scalability

Because CDNs are complex systems, the response time may depend on a large number of variables, and the dataset might comprise hundreds of millions of requests. To efficiently evaluate the *what-if* scenarios, WISE must address how to efficiently organize and utilize the dataset. In this section, we discuss our approach to these problems.

Curse of Dimensionality. As the number of dimensions (variables in the dataset) grow, exponentially more data is needed for similar accuracy of estimation. WISE uses the CBN to mitigate this problem. In a CBN, when conditioned on its parents, a variable is independent of all variables except its descendants. As a result we can use only the parents of the target variable for function estimation. Because the cardinality of the parent set would typically be less than the total number of variables in the dataset, the accuracy of the estimated function is significantly improved for a given amount of data. Due to this, WISE can afford to use fewer training data points and still get good accuracy.

Reducing dimensions also helps with computational complexity. For example, time complexity for the KR method is $\mathcal{O}(kn^3)$, with k variables and n points in the training dataset. Using a CBN reduces k and results in significant computational speedup.

Overfitting. The density of the dataset from a real deployment can be highly irregular; usually there are many points for combinations of variable values that represent the *normal* network operation, while the density of dataset is sparser for combinations that represent the fringe cases. Unfortunately, because the underlying principle of most regression techniques is to find parameters that minimize the errors on the training data, we can end up with parameters that minimize the error for high density regions of the dataset but give poor results in the fringes—this problem is called overfitting. The usual solution to this problem is introducing a smoothness penalty in the objective function of the regression method, but finding the right penalty function requires cross-validation, which is usually very expensive to compute for very large spaces.

WISE uses piece-wise regression to address this problem. WISE divides the dataset space into small buckets, that we refer to as *tiles*, and performs regression independently for each tile. Because regression parameters are learned separately for each of small buckets, it avoids the over-fitting problem. More details of tiles follows in context of approximate NN search.

Approximating Nearest-Neighbor Search. WISE uses nearest-neighbors (NN) search for estimating the distribution of variables after intervention. Unfortunately, NN search can be expensive: finding k nearest-neighbors for all the samples in a dataset is $\mathcal{O}(n^2)$. WISE overcomes this problem by limiting the scope of NN search. WISE divides the dataset space into *small* buckets, that we refer to as *tiles* and all the samples that lie in a bucket are considered equally likely nearest neighbors.

To create tiles, WISE uses fixed-size buckets for each dimension in the dataset.

Size of buckets presents a tradeoff: If the bucket sizes are sufficiently small, the samples in the bucket are good approximation of the NN, but the likelihood of having many samples lying in a bucket reduces with the size of the bucket. If the bucket is large, likelihood of finding samples in the bucket increases, but some of the points might be far away from the center of the bucket and they will not be good representation of NN. Fortunately, having more samples beyond a certain threshold does not contribute appreciably to the accuracy of estimating the empirical distribution of variables in question with either NN or KR techniques.

Keeping in view the tradeoff between bucket size and accuracy, and the need for limited training data samples per bucket, WISE uses a *multi layer tiling* scheme. For the lowest layer of tiles, WISE uses very small bucket sizes, and for subsequent highest layers, WISE increases the size of buckets. WISE processes the training data and maintains up to n_{max} sample points for each tile in each layer because more samples add to storage and lookup complexity but do not contribute to accuracy. When performing a NN search, WISE maps the test point to a tile in the lowest layer. If the tile has more than n_{min} samples, all the samples in the tile are considered the NN for the test sample. If there are fewer than n_{min} points in the tile, then WISE maps the test sample to a tile in next higher layer and repeats the process. If all the layers are exhausted but WISE is not able to find a suitable NN, the search aborts. In practice, two or three layers of tiles suffice for reasonable accuracy.

Parameters n_{min} and n_{max} are tunable. Our current implementation of WISE uses $n_{min} = 1$ for NN search and $n_{min} = 10$ for KR. n_{max} is set to 200.

Retrieval of Data. Because the datasets that WISE uses are very large, (often comprising several hundred million samples), rapid retrieval of data for evaluating the scenarios is critical. WISE expedites data retrieval by indexing the training dataset offline, and indexing the test dataset as it is generated during the scenario evaluation. Tiles are used here as well: Each tile is assigned a *tile-id*, which is simply a string formed by concatenating the tile's layer number and tile's boundaries in each dimension. Each data samples is assigned keys that are the tile-id of the tiles in which data samples lie. WISE then uses these keys to index the data samples. Because the WISE uses fixed size buckets, mapping a sample to its tile can be performed in constant time using simple arithmetic operations. Also, because the approximated NN search algorithm in WISE simply matches training and test data samples that share a tile, pre-indexing the data allows WISE to easily collate training and test data and perform NN search.

Parallelization and Batching. We have carefully designed various stages in WISE to support parallelization and batching of jobs that use similar or same data. In the training data preparation stage, each entry in the dataset can be independently assigned its *tile-id* based key because WISE uses fixed-sized tiles.

For NN search, WISE can find NN for all the test data samples that share a tile in one go because NN samples for one test sample are also NN for other test samples that share the tile. Further, the NN search for test points in different tiles can proceed in parallel, because NN search for each tile is independent.

For KR, WISE can learn the regression parameter for each tile independently and in parallel. Similarly, during response time prediction stage, WISE can perform prediction of test data belonging in separate tiles in parallel. Prediction for test data samples that share a tile is also batched because they all need same training data.

2.6.2 Missing Variables and Causal Relationships

False or missing causal relationships can occur if the population in the dataset is not independent of the outcome variables. Unfortunately, because WISE relies on passive datasets, this is a fundamental limitation that cannot be avoided. Fortunately, we expect that because the basic principles of computer networks are similar across the Internet, and the service providers use essentially the same versions of software throughout their networks, the bias in the dataset that would significantly affect the causal interpretation is not common. If such biases exist, they will likely be among datasets from different geographical deployment regions.

To catch such biases, we recommend using a training dataset with WISE that is obtained from different geographical locations. If we use this geographically diverse dataset to infer a causal structure, then if a variable that signifies the difference of datasets of two regions is missing in the dataset, then the variable representing the "region" will not be *separable* from the outcome variables. As a result, if we observe that there are variables in the DAG that are children of the "regional" variables, but there is no good explanation for this dependency based on understanding of the working of the system, then we have to investigate what the missing variables may be. Unfortunately, there is no automated way of determining what variables may be missing.

Lastly, it is difficult to compile a dataset that covers CDN system dynamics under *all* possible values of the variables in the system and from all the regions. As a result, if WISE learns the causal structure from a non-comprehensive dataset, then we cannot be certain that the causal structure captures all the variables and causal relationships. Such limitations can be detected by using dataset from one subset of geographical regions and use the dataset and causal structure to predict response time for another region and compare that with the ground truth. We present evaluation for completeness of causal structure and variables in Section 2.8.6. We also discuss a case where a missing variable in the dataset results in larger prediction error.

2.7 Implementation

We have implemented WISE with the Map-Reduce framework [16] using the Sawzall logs processing language [46] and Python Map-Reduce libraries. We chose this framework to best exploit the parallelization and batching opportunities offered by the

WISE design[1]. We have also implemented a fully-functional prototype for WISE using a Python front-end and a MySQL backend that can be used for small scale datasets. We provide a brief overview of the Map-Reduce based implementation here.

WISE steps are implemented using a combination of one or more of the four Map-Reduce patterns shown in Figure 2.7. These patterns are described below.

Filter pattern selects samples from the dataset that satisfy a certain criteria. In the input data preparation phase, the *use*-statement is implemented using the *filter* pattern.

Update pattern changes one or more fields in a subset of records in a dataset that satisfy a given criteria. *update*-statements use *update* pattern for applying the new values to the variable in the statement.

Training and Test Data Collation pattern aggregates training and test data that share a tile. If the *update*-statement uses the INTERVENE keyword then WISE uses the *Training & Test Data Collation* pattern to bring together the relevant test and training data and update the distribution of the test data in a batched manner and perform KR or NN search.

Tile-id Assignment pattern indexes the training and test data samples. This pattern is used on the initial training and test dataset as well on the test dataset after each *update*- statement because the changes in the value of the data may necessitate re-assignment of the tile-id.

Our Map-Reduce based implementation can evaluate typical scenarios in about 10 minutes on a cluster of 50 PCs while using nearly 1 TB of training data.

[1]Hadoop[1] provides an open-source Map-Reduce library. Modern data-warehousing appliances, such the ones by Netezza [43], can also exploit the parallelization in WISE design.

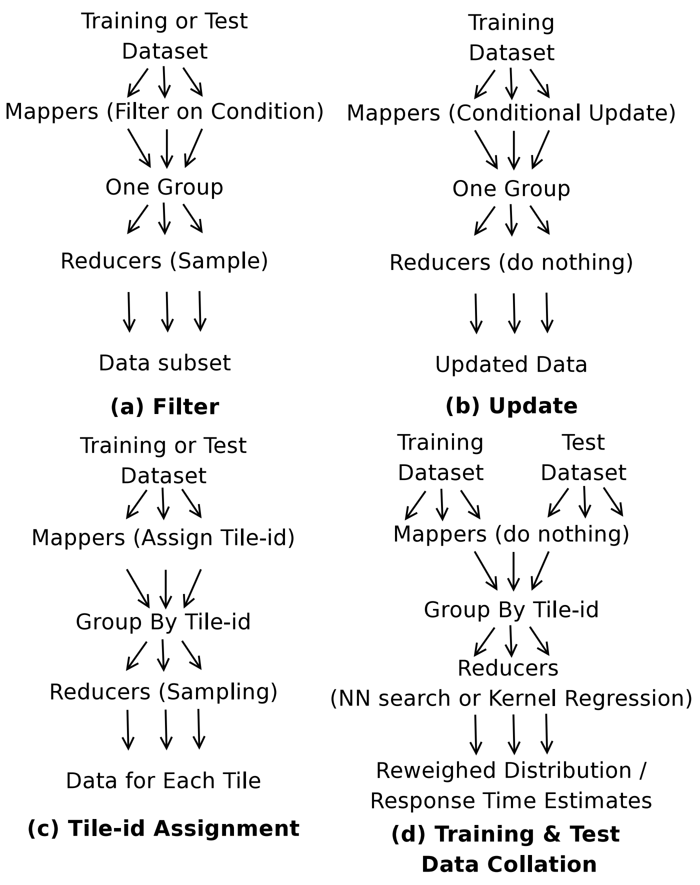


Figure 7: Map-Reduce patterns used in WISE implementation.

2.8 Evaluating WISE on a Deployed CDN

We have implemented WISE with the Map-Reduce framework [16] using the Sawzall logs processing language [46] and Python Map-Reduce libraries. We chose this framework to best exploit the parallelization and batching opportunities offered by the WISE design[2]. We have also implemented a fully-functional prototype for WISE using a Python front-end and a MySQL backend that can be used for small scale datasets. We provide a brief overview of the Map-Reduce based implementation here.

In this section, we describe our experience applying WISE to a large dataset

[2]Hadoop[1] provides an open-source Map-Reduce library. Modern data-warehousing appliances, such the ones by Netezza [43], can also exploit the parallelization in WISE design.

obtained from Google's global CDN for Web-search service. We use WISE to predict two metrics for this service:

- Network-level server response time (*nrt*) is the time duration between client's host sending a HTTP request with Web-search query and the client receiving the last byte of the response from the CDN for Web search.

- Browser-level response time (*brt*) is the time duration between the user clicking the "Search" button on the Web page and when the browser completes loading the results pages returned by the CDN for Web search.

We start by briefly describing the CDN and the service architecture in Section 2.8.1. In Section 2.8.2 we describe the dataset from the CDN. In Section 2.8.3 we describe the factors that make it challenging to estimate *nrt* and *brt* using this dataset. In Section 2.8.4 the causal structure discovered from this dataset using WCD. In Sections 2.8.5—2.8.7 we evaluate WISE's ability to predict response-time distribution for the *what-if* scenarios.

2.8.1 Web-search Service Architecture

Figure 2.8.1(a) shows Google's Web-search service architecture. The service comprises a system of globally distributed HTTP reverse proxies, referred to as front-end (FE), and a system of globally distributed clusters that house the Web servers and other core services, referred to as the back-end (BE). A DNS based request redirection system redirects the user's queries to one of the FE in the CDN. The FE process forwards the queries to the BE servers, which generate dynamic content based on the query. The FE caches static portions of typical reply, and starts transmitting that part to the requesting user as it waits for reply from the BE. Once the BE replies, the dynamic content is also transmitted to the user. The FE servers may or may not be colocated in the same data center with the BE servers. If they are colocated, they can be considered to be on the same local area network and the round-trip latency between them is only a few milliseconds. Otherwise, the connectivity between the FE and the BE is typically on a well-provisioned connection on the public Internet. In this case the latency between the FE and BE can be several hundred milliseconds.

Network-level server response time (nrt) for a request is the time between the instance when the user issues the HTTP request and the instance when the last byte of the response is received by the users. We estimate *nrt* as the sum of the round-trip time estimate obtained from the TCP three-way handshake, and the time between the instance when the request is received at the FE and when the last byte of the response is sent by the FE to user. Key contributors to *nrt* are: (i) the transfer latency of the request from the user to the FE, (ii) the transfer latency of request to the BE and the transfer latency of sending the response from the BE to the FE, (iii) processing time at the BE, (iv) TCP transfer latency of the response from the FE to the client; and (v) any latency induced by loss and retransmission of TCP segments.

Browser-level service response time (brt) for a request is the time between the instance when the user presses the "Search" button on the Web page and when the browser completes rendering the Web page with search results that the server sends. We estimate *brt* using a browser plug-in and javascript embedded in the Web page returned by the server. Following are the factors contributing latency to *brt*. (i) User's host may issue a DNS lookup before it can establish a connection with the Web server. When this happens, DNS lookup delay contributes to *brt*. (ii) Network-level server response captures the latency in host contacting the servers, servers preparing a search response and sending the response to the client. (iii) Time it takes the browser to render the results Web page and execute javascripts embedded in the page.

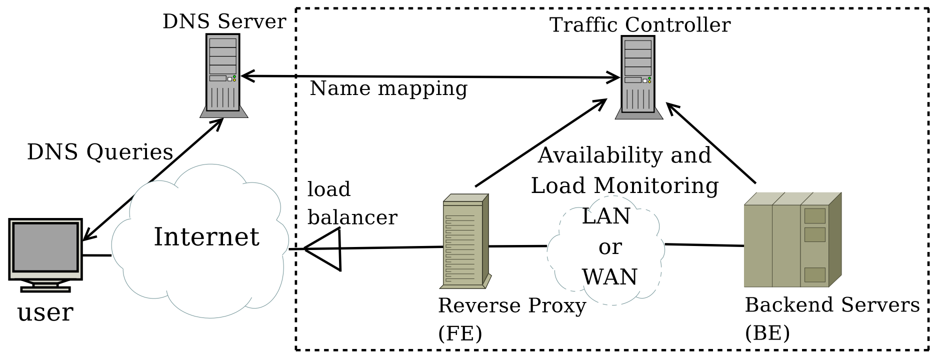


Figure 8: Service architecture for Google's Web Search service.

2.8.2 Dataset

We use data from an existing network monitoring infrastructure in Google's network. FE and BE servers export reports with values for many performance related variables. A browser plug-in tracks browser-level events and reports these statistics for users that opt to participate in monitoring. The browser plug-in and FE add unique identifiers to each request. This identifier allows us to collate the records in dataset obtained from FE, BE, and the browser plug-ins.

WISE applies the feature selection tests (ref. Section. 2.5.1) on the variables in the dataset. Table 1 describes the variables that WISE found to be relevant to network and browser response-time variables.

2.8.3 Challenges in Estimating nrt and brt

Figure 2.8.1 shows the process by which a user's Web search query is serviced. The message exchange and events has many factors that affect *nrt* and *brt* in subtle ways, making it hard to make accurate "back-of-the-envelop" calculations in the general case.

Challenges for Network Response Time (*nrt*). Following factors make it difficult to estimate *nrt*.

Asynchronous transfer of content to the user. Once the TCP handshake is complete,

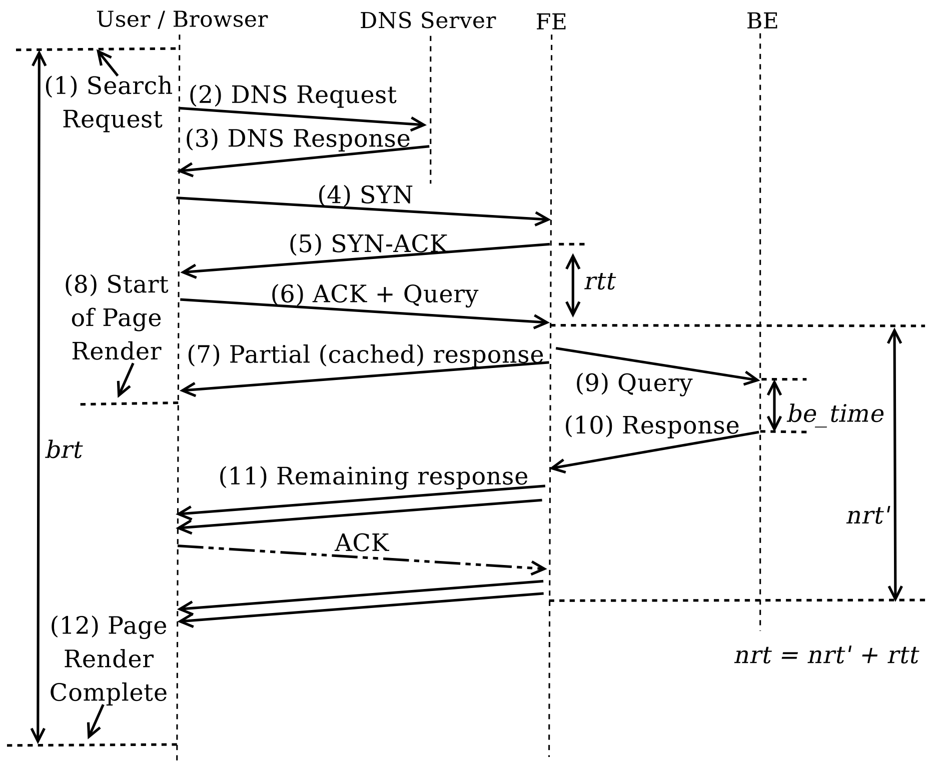


Figure 9: Messages and events for network-level and browser-level response time.

user's browser sends an HTTP request containing the query to the FE. While the FE waits on a reply from the BE, it sends some static content to the user; this content—essentially a "head start" on the transfer—is typically brief and constitutes only a couple of IP packets. Once the FE receives the response from the BE, it sends the response to the client and completes the request. A client may use the same TCP connection for subsequent HTTP requests.

Spliced TCP connections. FE processes maintain several TCP connections with the BE servers and reuse these connections for forwarding user requests to the BE. FE also supports HTTP pipelining, allowing the user to have multiple pending HTTP requests on the same TCP connection.

Spurious retransmissions and timeouts. Because most Web requests are short TCP transfers, the duration of the connection is not sufficient to estimate a good value for

Table 1: Features in the dataset from Google's CDN.

Feature	**Description**
ts, tod	A timestamp of instance of arrival of the request at the FE. Hourly time-of-day (tod) is derived from the timestamp.
sP	Number of packets sent by the server to the client excluding retransmissions.
srP	Number of packets retransmitted by the server to the client, either due to loss, reordering, or timeouts at the server.
sB	Size in bytes of the encoded response sent by the FE server.
encoding	Encoding type used by FE server.
region	User's IP address, /16, /24 network prefixes, AS number, and geographical mapping to state and country, are collectively referred to as *region*.
fe, **be**	Identifiers for the FE data center at which the request was received and the BE data center that served the request.
rtt	Round-trip time between the user and FE estimated from the initial TCP three-way handshake.
bw	An estimate of access network bandwidth for the /24 IP address block for the user's IP address. It is estimated as the 90% TCP throughput for unthrottled responses greater than 4KB sent from FE to the IP addresses in /24 prefix.
febe_rtt	The network level round-trip time between the front end and the back-end clusters.
be_time	Time taken by BE to process the request forwarded by FE.
firstReq	Binary variable indicating whether the HTTP request is first on a TCP connection.
browser	Browser name and version obtained from HTTP User-Agent string.
nrt	Network-level response time for the request as seen by the FE (see Section 2.8.1). *nrt* is a target variable.
brt	Browser-level response time for the request as seen by the FE (see Section 2.8.1). *brt* is a target variable.

the TCP retransmit timer and many Web servers use default values for retransmits, or estimate the timeout value from the initial TCP handshake round-trip time. This causes spurious retransmits for users with slow access links and high serialization delays for MTU sized packets.

Challenges for Browser Response Time (*brt*). Following factors make it difficult to estimate *brt*.

DNS lookup not visible to network or browser. User's host needs to resolve the domain name for the search engine site before it can contact the servers. If the DNS reply is cached at the client, latency for DNS resolution is minimal, however, if DNS reply is not cached or has expired, then DNS latency can be significant. Therefore, not all search requests include DNS resolution latency. Because DNS lookup is transparent to the browser, we have no data to indicate the latency for DNS for a particular request request at browser level. Further, because clients typically use local DNS resolvers, the DNS request is not seen by the DNS servers of the CDN either. As a result, our data has no direct indication of latency contributed by DNS.

Browser may fetch additional resources. Web page rendering in the browser is only complete after the browser has rendered the HTML and other embedded content, such as, images or scripts, referred in the HTML page. This content is often cacheable, but if it not cached at the host or has expired, then the browser may need to fetch this content before it can complete rendering. Unfortunately, to encourage caching, the URLs for this content are generic and non-collatable with the main HTTP request for Web search. As a result, our data has no direct indication of latency contributed by fetching of embedded content.

Browser type and user host's computation capability. Web page rendering time varies with the type of browser. We can infer the browser type from the HTTP User-Agent string, but relationship of rendering speed and browser type is not straightforward. Rendering time also depends on type of content (HTML, javascript, images), type of compression or encoding used in HTTP, the order of download of content, the computation capability of the host and the load on the host at the time of search query. Not only is host's computation capability and load not observed in the dataset, the relationship of these factors is not well understood to allow simple back-of-the-envelop calculations.

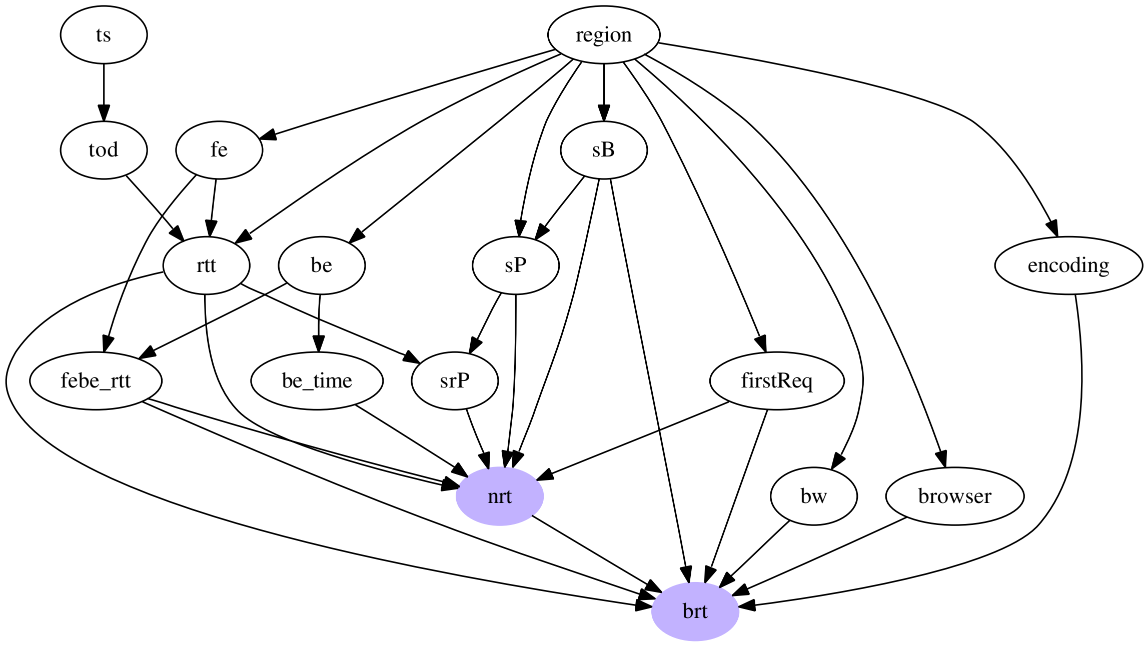


Figure 10: Inferred causal structure in the dataset. $A \rightarrow B$ means A *causes* B. Target variables are shown in shaded nodes.

2.8.4 Causal Structure in the Dataset

To obtain the causal structure, we use a small sampled data subset collected in July, 2009, from several data center locations. This dataset has roughly 50 million requests, from clients in more than 10000 unique ASes.

We seed the WCD algorithm with the *region* and *ts* variables as the *no-cause* variables. Figure 2.8.4 shows the causal structure that WCD produces. Most of the causal relationships in Figure 2.8.4 are straightforward and make intuitive sense in the context of networking, but a few relationships are quite surprising. WCD detects a relationship between the *region* and *sB* attribute (the size of the response from the server); we found that this relationship exists due to the differences in the sizes of search response pages in different languages and regions as well as the encoding and compression schemes used by the servers. Another unexpected relationship is between *region* and *sP* attributes; We suspect that this relationship exists due to different MTU sizes in different parts of the world. Unfortunately, our dataset did not have load, utilization, or data center capacity variables that could have allowed us to model the *be_time* variable. All we observed was that the *be_time* distribution varied somewhat among the data centers. Overall, we find that WCD algorithm not only discovers relationships that are faithful to how networks operate but also discovers relationships that might escape trained network engineers.

Crucially, note that many variables, including the target variables (*nrt, brt*) are not direct children of the *region*, ts, fe, or *be* variables. This means that when conditioned on their respective parents, these variables are independent of the region, time, choice of FE and BE, and we can use training data from past, different regions, and different FE and BE data centers to estimate the distributions for these features! Further, while most of the variables in the dataset are correlated, the in-degree for each variable is smaller than the total number of variables. This reduces the number of dimensions that WISE must consider for estimating the value of the variables during scenario evaluation, allowing WISE to produce accurate estimates, more quickly and with less data.

2.8.5 Estimation of Response Time

Our primary metric for evaluation is *prediction accuracy*. There are two sources of error in response-time prediction: (i) error in response-time estimation function (Section 2.5.5) and (ii) inaccurate input, or error in estimating a valid input distribution that is representative of the scenario (Section 2.5.4). To isolate these errors, we first evaluate the estimation function accuracy alone in Section 2.8.6 and later consider the overall accuracy for a complete scenario in Section 2.8.7.

2.8.6 Accuracy of Response-time Function

Accuracy of the response-time function estimator depends on following: (a) whether the WISE model captures all the variables, (b) whether the causal structure that

WISE infers is accurate, and (c) whether there is sufficient training data available to make the prediction. In this section, we evaluate whether these factors are reasonably addressed by WISE by estimating the accuracy of predictions of (*nrt, brt*) for existing deployments.

To evaluate prediction for existing deployments we can try to evaluate a scenario: "What-if I make no changes to the network?" This scenario is easy to specify with WSL by not including any optional scenario update statements. For example, a scenario specification with the following line: `USE WHERE country==deu` would produce an input distribution for the response-time estimation function that is representative of users in Germany without any error and any inaccuracies that arise would be due to inaccurate input from scenario specification.

Recall from Sections 2.5.4 and 2.5.5 that WISE uses the parents of *nrt* and *brt* to find the nearest-neighbor samples from the training dataset and predict the response time. Region, FE, BE, or *ts* are not parents of CBN in Figure 2.8.4. As a result, when WISE performs NN search, it does so without regard for the region, servers, or timestamp of the samples. Still, to ensure that the nearest-neighbor samples that WISE uses for prediction do not come from same data center or country for which we wish to evaluate a *what-if* scenario, we exclude the data from those data centers and countries from the training dataset for all of the evaluations that follow.

Network-level Response Time (*nrt*) Prediction. We start with demonstrating that WISE predicts entire response time distribution accurately. We then compare WISE predictions for *nrt* with simpler parametric models.

Predicting Distribution of nrt. We examine accuracy of *nrt* prediction for three scenarios. First two scenarios specify estimating the response-time distribution for the users in Germany, and South Africa and the third scenario tries to estimate the response-time distribution for users that are served from FE in Japan. This FE data center primarily serves users in South and East Asia. We used the dataset from the third week of June 2007 as our training dataset and predicted the network response time distribution for these scenarios for the fourth week of June 2007.

Figures 11(a), (b), and (c), show the results for the three scenarios, respectively. The ground-truth distribution for response time is based on the response-time values observed by the monitoring infrastructure for fourth week of June 2007, and is shown in solid line. The response time for the same period that WISE predicts is shown in a dotted line. The ground-truth and predicted distributions are identical.

WISE also predicts response time for individual requests accurately. Figure 12 we present the relative prediction error for individual requests in these experiments. The error is defined as $|rt - \widehat{rt}|/rt$, where rt is the ground-truth value and $\widehat{rt}$ is the value that WISE predicts. The median error lies between 8-11%.

Comparing WISE with Parametric Models. Because Google uses TCP to transfer the data to clients, It is reasonable to ask how well we could do using one of simpler parametric models used for estimating TCP-transfer latency? We have adapted work of Arlitt et al. [4] to account for additional latency that occurs due to round-trip between FE and BE (`febe_rtt`) and the back-end processing time (`be_time`). We refer to this model as AKM.

Table 2 presents a comparison of relative error in mean network response time estimates for ten regions using WISE and AKM. Table 2-a presents errors for cases where the serving FE and BE are colocated in the same data center. Table 2-b presents errors for cases where serving FE not in the same data center as the BE. These are also cases where access network is constrained, and there are significant packet losses and retransmissions. The relative error for mean response time for colocated cases is comparable between WISE and AKM. For non-colocated cases, error for AKM is between 3.5% and 9% more than WISE.

AKM uses a compensation multiplier called *'CompWeight'* to minimize the error

Table 2: Comparison of WISE and parametric approach based on relative prediction error.

(a) Co-located FE & BE			(b) Non-colocated FE & BE		
Region	WISE	AKM	Region	WISE	AKM
Belgium	2.0%	1.1%	Argentina	-2.2%	-11%
Germany	0.3%	1.0%	Australia	-0.9%	-10%
Hungary	0.3%	-6.7%	Brazil	0.2%	-8.9%
Netherlands	0.0%	4.4%	Hong Kong	-3.0%	-7.7%
USA	1.4%	1.1%	Singapore	2.2%	5.9%

for each trace. This factor minimizes the error on the mean at cost of higher error in the tails of the distribution. WISE, on the other hand, can predict the entire distribution accurately and also predicts the mean response time more accurately than AKM. Further WISE has advantage of generality. As we show in next subsection, WISE is easily extended to predict browser-level response time.

Browser-level Response Time (brt) Prediction. Browser-level response time is a high variance metric because of nature of Web page rendering process. For a typical region, both mean and standard deviation of *brt* is between 2 to 3 times more than mean and standard deviation for *nrt*. Further, as discussed in Section 2.8.3, our dataset does not contain variables that capture important contributing factors such as, DNS latency, how many embedded pieces of contents are required to render the page, whether the browser downloaded the embedded content or obtained these from cache, the computation capability of the user's host and the load on that host at the time of request. This makes estimating browser-level response time significantly more difficult.

We use WISE to estimate the browser level response time distribution for seven regions, served from a mix of colocated and non-colocated FE and BE locations. We use datasets from first two weeks in September 2009. Table 3 presents the relative error on key percentiles of the distribution of *brt* as well as mean *brt* for these regions.

Table 3: Relative error of mean browser-level response time estimate with WISE.

Region	Error on Percentiles			Error on
	25th	50th	75th	Mean
B	12.0%	3.6%	4.1%	4.8%
C	1.7%	-3.9%	-10.1%	4.1%
D	0.4%	-1.6%	-4.2%	-10.9%
E	-1.2%	-3.6%	-5.2%	4.7%
F	0.3%	-1.5%	-2.5%	4.1%
G	9.2%	9.9%	10.7%	11.6%
I	4.0%	4.8%	5.8%	7.7%
D*	-3.1%	-2.6%	-3.4%	-7.8%
G*	1.4%	1.9%	4.4%	5.5%

Prediction error on all key percentiles as well on mean for most regions is less than 10%. This is remarkable considering that *brt* is a very volatile metric and the dataset is missing indicators for a number of contributors of *brt* latency.

For two regions, D and G, the relative error for mean *brt* is greater than 10%. By examining javascript execution script in a limited scope experiment, we found that the reason for larger error is that the browser and computation capability combination in these countries is pretty unique, but because there is no feature in the dataset to indicate computation capability of hosts, the nearest-neighbor search algorithm in WISE ends up using samples that are similar in other dimensions but differing in computation capability, resulting in higher prediction error. We further ascertained this observation by using data from the same regions as training data. The results are shown at the bottom of Table 3 as D* and G*. When we use the local datasets, prediction error drops significantly.

We are working to extend our datasets to include features that are indicative or computation capability of the hosts. Once these variables are available, we believe we will be able to improve prediction accuracy without using local datasets.

2.8.7 Evaluating a Live What-if Scenario

We evaluate how WISE predicts the response-time distribution for the affected set of customers during the scenario that we presented in Section 2.2 as a motivating example. In particular, we will focus on the effect of this event on customers of AS 9498, which is a large consumer ISP in India.

To appreciate the complexity of this scenario, consider what happens on the ground during this reconfiguration. First, because the FE in Taiwan is colocated with the BE, $febe\_rtt$ reduces to a typical intra-data center round-trip latency of 3ms. Also we observed that the average latency to the FE for the customers of AS 9498 increased by about 135ms as they were served from FE in Taiwan (`tw`) instead of the FE in India (`im`).

If the training dataset already contains the rtt estimates for customers in AS 9498 to the fe in Taiwan then we can write the scenario in two lines as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET fe=tw
```

WISE uses the CBN to automatically update the scenario distribution. Because, fe variable is changed, WISE updates the distribution of children of fe variable, which in this case include $febe\_rtt$ and rtt variables. This in-turn causes a change in children of rtt variable, and similarly, the change cascades down to the rt variable in the DAG. In the case when such rtt is not included in the training dataset, the value can be explicitly provided as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET febe_rtt=3
INTERVENE SET rtt+=135
```

We evaluated the scenario using the former specification. Figure 13 shows ground truth of response-time distribution as well as distributions for intermediary variables

(Figure 2.8.4) for users in AS 9498 between hours of 12 a.m. and 8 p.m. on July 16th and the same hours on July 17th, as well as the distribution estimated with WISE for July 17th for these variables. We observe only slight under-estimation of the distributions with WISE—this underestimation primarily arises due to insufficient training data to evaluate the variable distribution for the peripheries of the input distribution; WISE was not able to predict the response time for roughly 2% of the requests in the input distribution. Overall, maximum cumulative distribution differences[3] for the distributions of the three variables were between 7-9%.

2.9 Controlled Experiments

Because evaluating WISE on a live production network is limited by the nature of available datasets and variety of events with which we can corroborate, we have created a small-scale Web service environment using the Emulab testbed [19]. The environment comprises a host running the Apache Web server as the back-end or BE, a host running the Squid Web proxy server as the front-end or FE, and a host running a multi-threaded `wget` HTTP client that issues request at an exponentially distributed inter-arrival time of 50 ms. The setup also includes delay nodes that use dummynet to control latency.

To emulate realistic conditions, we use a one-day trace from several data centers in Google's CDN that serve users in the USA. We configured the resource size distribution on the BE, as well as to emulate the wide area round-trip time on the delay node based on this trace.

For each experiment we collect `tcpdump` data and process it to extract a feature set similar to one described in Table 1. We do not specifically emulate losses or retransmits because these occurred for fewer than 1% of requests in the USA trace.

[3]We could not use the relative error metric here because the requests in the input distribution prepared with WISE for *what-if* scenario cannot be pair-wise matched with ones in the ground-truth distribution; maximum distribution difference is a common metric used in statistical tests, such as Kolmogorov-Smirnov Goodness-of-Fit Test.

We have conducted two *what-if* scenario experiments in this environment; these are described below.

Experiment 1: Changing the Resource Size. For this experiment, we used only the back-end server and the client machine, and used the delay node to emulate wide area network delays. We initially collected data using the resource size distribution based on the real trace for about two hours, and used this dataset as the training dataset. For the *what-if* scenario, we replaced all the resources on the server with resources that are half the size, and collected the test dataset for another two hours. We evaluated the test case with WISE using the following specification:

```
USE *

INTERVENE SET FIXED sB/=2
```

Figure 14(b) presents the response-time distribution for the original page size (dashed), the observed response-time distribution with halved page sizes (dotted), and the response-time distribution for the *what-if* scenario predicted with WISE using the original page size based dataset as input (solid). The maximum CDF distance in this case is only 4.7%, which occurs around the 40^{th} percentile.

Experiment 2: Changing the Cache Hit Ratio. For this experiment, we introduced a host running a Squid proxy server to the network and configured the proxy to cache 10% of the resources uniformly at random. There is a delay node between the client and the proxy as well as the proxy and the back-end server, each emulates trace driven latency as in the previous experiment. For the *what-if* scenario, we configured the Squid proxy to cache 50% of the resources uniformly at random. To evaluate this scenario, we include binary variable `b_cached` for each entry in the dataset that indicates whether the request was served by the caching proxy server or not. We use about 3 hours of trace with 10% caching as the training dataset, and use WISE to predict the response-time distribution for the case with 50% caching by using the following specification:

```
USE *
INTERVENE SETDIST b_cached FILE 50pcdist.txt
```

The `SETDIST` directive tells WISE to update the `b_cached` variable by randomly drawing from the empirical distribution specified in the file, which in this case contains 50% 1s and 50% 0s. Consequently, we intervene 50% of the requests to have a cached response.

Figure 14(c) shows the response-time distribution for the 10% cache-hit ratio (dashed), the response-time distribution with 50% cache-hit ratio (dotted), and the response-time distribution for the 50% caching predicted with WISE using the original 10% cache-hit ratio based dataset as input (solid). WISE predicts the response time quite well for up to the 80^{th} percentile, but there is some deviation for the higher percentiles. This occurred because the training dataset did not contain sufficient data for some of the very large resources or large network delays. The maximum CDF distance in this case is 4.9%, which occurs around 79^{th} percentile.

2.10 Discussion

In this section, we discuss the limitations and extensions for WISE. First we discuss the limitations in terms of scenarios that WISE can predict. Then we present the difficulties that arise for specifying hypothetical values for network round-trip times for evaluating server placement or peering scenarios, and our on-going work to address these. We discuss merit of using non-parametric modeling in WISE, and finally, how the WISE framework can be extended to other realms in networking.

What Can and Cannot Be Predicted? The class of *what-if* scenarios that can be evaluated with WISE depends entirely on the dataset that is available; in particular, WISE has two requirements:

First, WISE requires expressing the *what-if* scenario in terms of (1) variables in the dataset and (2) manipulation of those variables. At times, it is possible for dataset to capture the effect of the variable without capturing the variable itself. In such cases, WISE cannot evaluate any scenarios that require manipulation of that hidden variable. For example, the dataset from Google, presented earlier, does not include the TCP timeout variable even though this variable has an effect on response time. Consequently, WISE cannot evaluate a scenario that manipulates the TCP timeout.

Second, WISE also requires that the dataset contains values of variables that are similar to the values that represent the what-if scenario. If the global dataset does not have sufficient points in the space where the manipulated values of the variables lie, the prediction accuracy is affected, and WISE raises warnings during scenario evaluation.

WISE also makes stability assumptions, i.e., the causal dependencies remain unchanged under any values of intervention, and the underlying behavior of the system that determines the response times does not change. We believe that this assumption is reasonable as long as the fundamental protocols and methods that are used in the network do not change.

Values of Variables for Scenario Specification. To predict response-time distribution, WISE requires the designers to specify the *what-if* value of one or more variables. While this considerably facilitates scenario specification and evaluation, it is still a big problem in some cases. Specifically, specifying a scenario with correct value of network round-trip time (RTT) distribution to evaluate a new peering, or data center deployment has proven to be difficult.

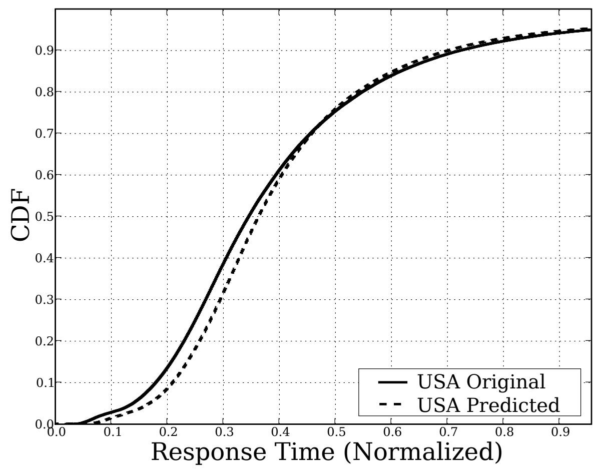
Predicting the RTT distribution for a change in the network is difficult for a many reasons: A new peering or transit relationship can affect BGP level routing for many AS. WISE in its current form, does not try to predict the AS whose traffic would be affected by the change. Further, even when the subset of traffic that would be affected is identified, the change in RTT is usually not a simple step-function: the changes in RTT value different at different percentiles of distributions, higher percentiles are usually affected less by a shorter network path than lower percentiles.

WSL allows the designers to specify full distribution of a RTT using the `SETDIST` directive, if the distribution is known. We are separately investigating on two aspects that would facilitated this process further. (1) We are working modeling of RTT distribution by separately modeling the propagation and queuing delay factors in RTT and their interaction. This separation will allow the designers to specify the changes in the propagation delay and WISE would automatically estimate the likely queuing delay component and estimate the RTT distribution more accurately. (2) By analyzing the the Internet connectivity graph, we are trying to estimate the propagation delays between AS and target CDN node locations. This would allow the designers to simply specify a hypothetical location for CDN node and WISE will be able to determine the propagation delays, queuing delays and the RTT distribution automatically. This RTT distribution can then be used to estimate overall response-time for the service.

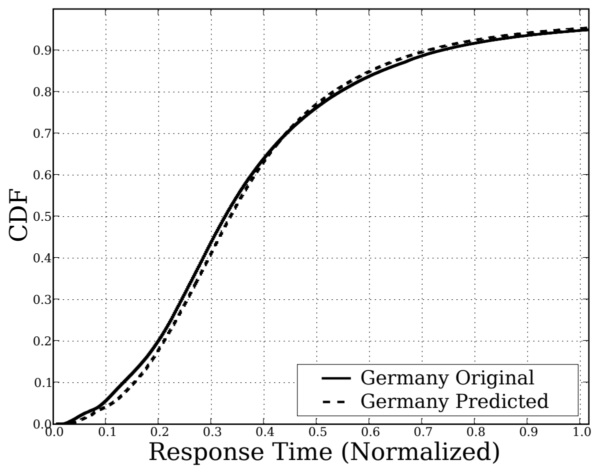
Parametric vs. Non-Parametric. WISE uses the assumption of functional dependency among variables to update the values for the variables during the statistical intervention evaluation process. In the present implementation, WISE only relies on non-parametric, nearest-neighbor search based technique for estimating this function. This makes it difficult to use WISE for cases where dataset may not have suitable nearest-neighbor samples. Nearest-neighbor based prediction is also data intensive. Fortunately, nothing in the WISE framework prevents using parametric functions. If the dependencies among some or all of the variables are parametric or deterministic, then we can improve WISE's utility. Such a situation can in some cases allow *extrapolation* to predict variable values outside of what has been observed in the training dataset.

What-if Scenarios in other Realms of Networking. We believe that our work on evaluating what-if scenarios can be extended to incorporate other realms, such as routing, policy decisions, and security configurations by augmenting the reasoning systems with a decision evaluation system, such as WISE.

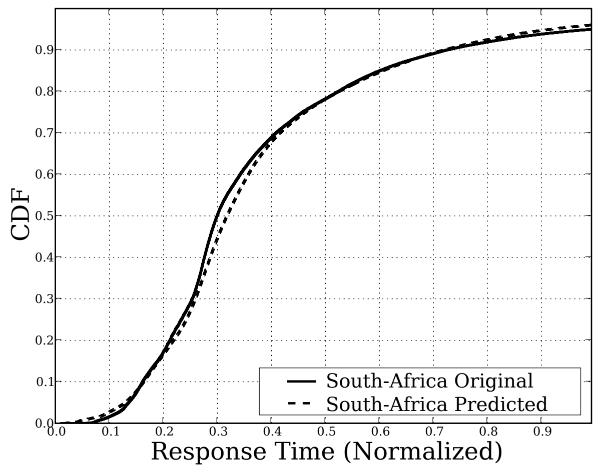
Our ultimate goal is to evaluate what-if scenarios for high-level goals, such as, "What if I deploy a new server at location X?", or better yet, "How should I configure my network to achieve certain goal?"; we believe that WISE is an important step in this direction.



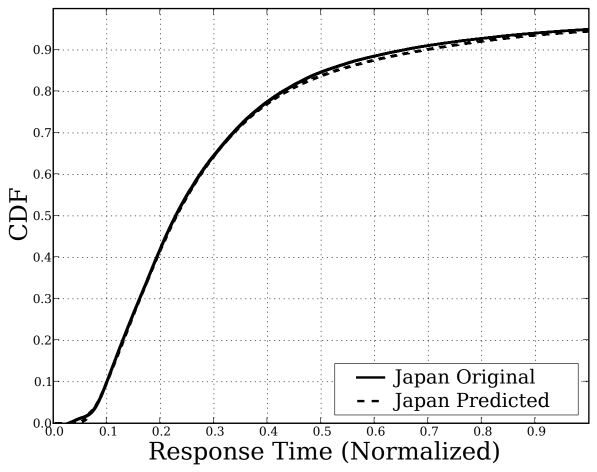
(a) USA



(b) Germany



(c) South Africa



(d) FE in Japan

Figure 11: Prediction accuracy: comparison of normalized network-level response time distributions for the scenarios in Section 2.8.6.

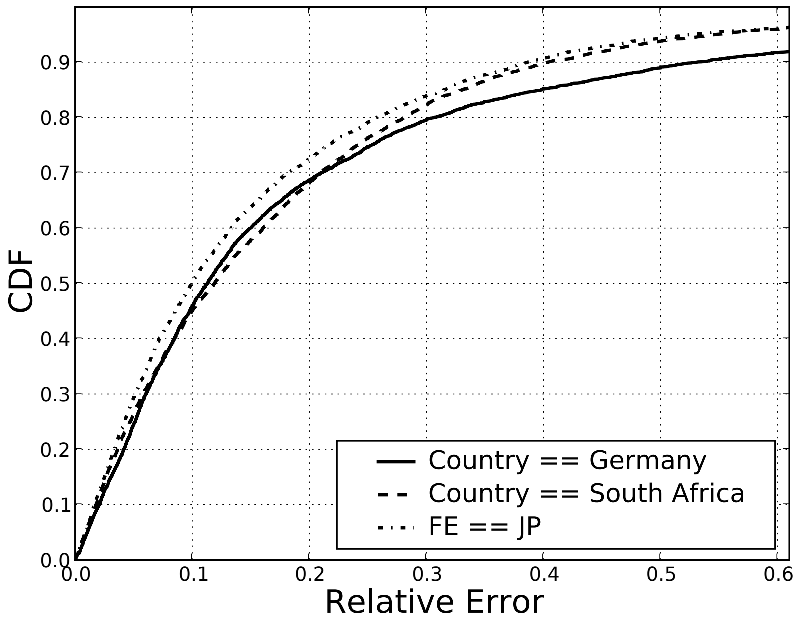
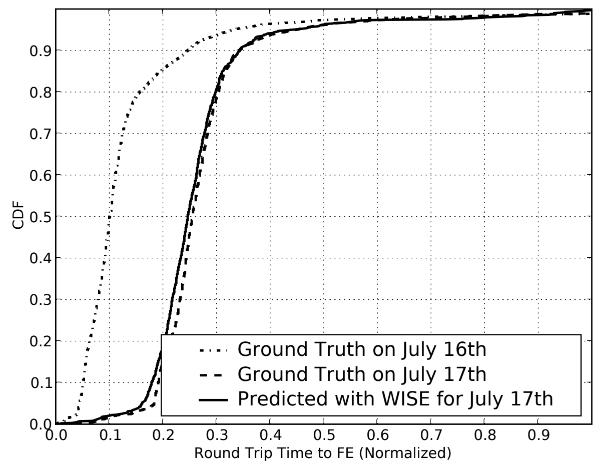
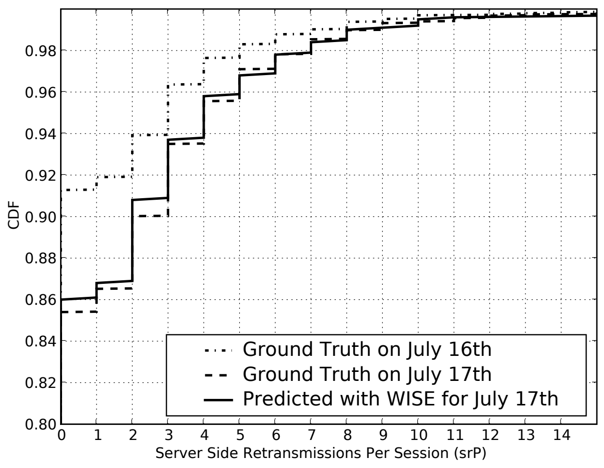


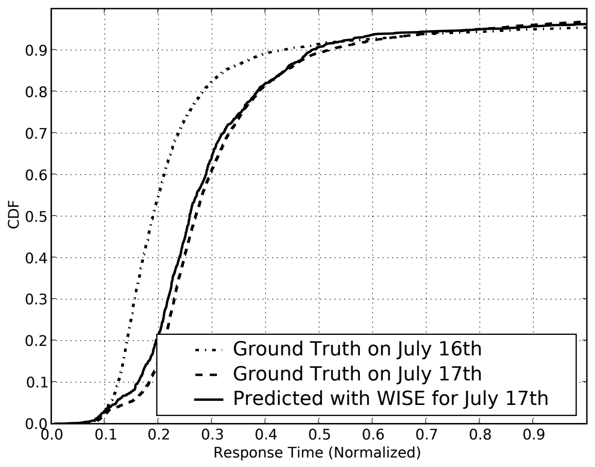
Figure 12: Relative prediction error for the scenarios in Figure 11.



(a) Client-FE RTT

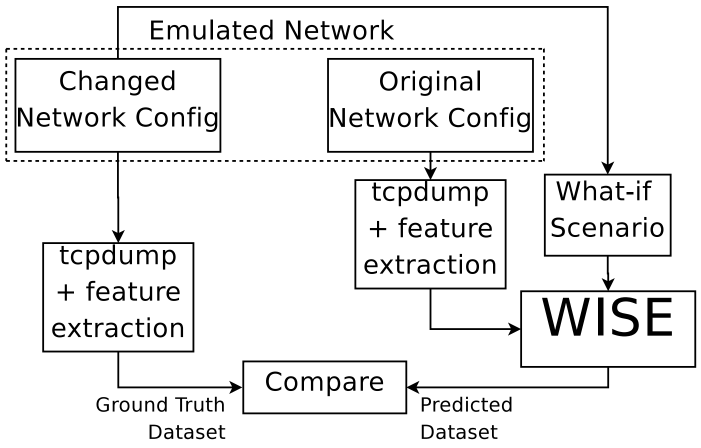


(b) Server Side Retransmits

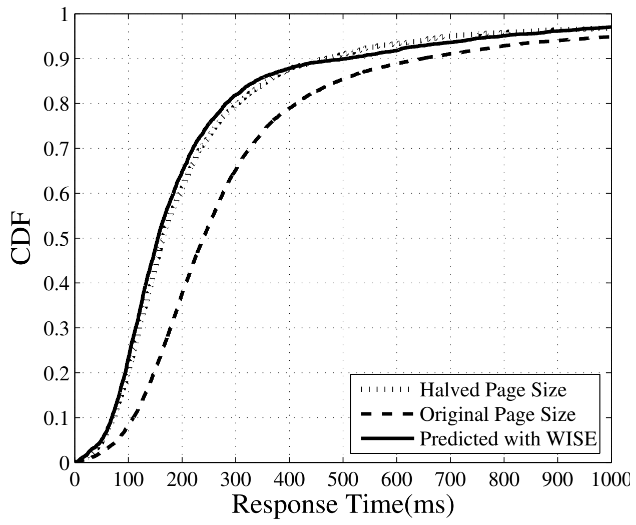


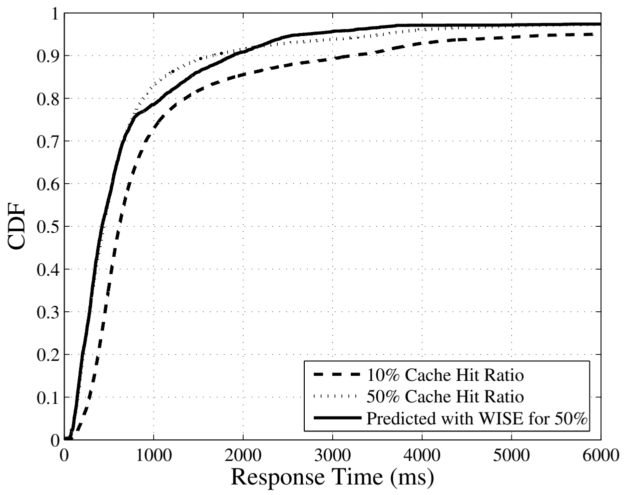
(c) Response Time Distribution

Figure 13: Predicted distribution for response time and intermediary variables for the India data center drain scenario.



(a) Controlled Experiment Setup





(b) Changing the Page Size

(c) Changing the Cache Hit Ratio

Figure 14: Controlled *what-if* scenarios on Emulab testbed: Experiment setup and results.

CHAPTER III

ANSWERING "HOW-TO" QUESTIONS FOR MITIGATING HIGH-LATENCY WEB TRANSACTIONS WITH HIP

ABSTRACT

This paper presents *How to Improve Performance* (HIP), a tool that facilitates answering *how-to* questions for mitigating high-latency Web transactions in content distribution networks (CDNs). Answering these questions is challenging because many factors affect service response time but their effect is difficult to quantify. Although recent tools such as What-if Scenario Evaluator (WISE) help designers predict the response times that result from a configuration, the number of possible configuration or deployment changes to a CDN is still so large that it is infeasible to use these tools "off the shelf" to search for a configuration that achieves the desired performance. To manage this search, HIP analyzes dependency among latency-causing factors to identify both the immediate and indirect causes of high latency transactions, segregates transient from persistent causes, and groups transactions that experience high latency due to the same underlying cause. Based on causes and affected groups of transactions, HIP suggests configuration changes that improve performance. We apply HIP to the Web search service from a large content provider. HIP finds that in most regions, 90% of high-latency incidents experienced by users can be explained by no more than three causes, which narrows the search for good configurations. We evaluate HIP's suggested configuration changes using WISE and find that they are effective in mitigating high-latency transactions.

3.1 Introduction

The performance that users experience for Web-based services depends on many interrelated factors. The time it takes between a user first making a request to a Web-based service and the response loading and rendering in user's browser application depends on the geographical location of the user and the server, proxy servers, the backend servers, properties of the content requested by the user, the time of day and load on the servers or network, the network bandwidth and round-trip time, packet loss rate, the capabilities of user's host machine, and applications that the user used to access the service. Web-based services are typically hosted on content distribution networks (CDNs) that have hundreds to thousands of distributed servers in tens of data centers [2, 7, 21].

Operators of these networks continually strive to improve the response times for Web-based services, which requires tackling questions such as, *How to improve overall performance by reconfiguring or re-provisioning the network?*, or *How to improve some subset of transactions that has higher latency.* We refer to these questions as "*how-to*" questions.

Answering *how-to* questions is difficult for several reasons. First, the factors that affect performance, and the interaction among these factors—and, in particular, how they affect response time—is not understood. Recent systems such as WISE [54] solve part of this problem by allowing CDN operators to evaluate the effect of configuration or deployment changes on response time. Thus, a strawman approach to answering *how-to* questions is to explore candidate *what-if* scenarios that may improve the performance or subset of users. Unfortunately, because the relationships between factors that affect the response time are poorly understood, finding good candidate scenarios in the first place is challenging. The operator may use brute-force search on many candidate scenarios, but the large search space may make this approach infeasible.

This chapter presents the design, implementation, and evaluation of *How to Improve Performance* (HIP), a tool that helps operators find configuration scenarios that answer the *how-to* questions. Instead of using brute force to find scenarios that can achieve desired performance, HIP first searches for common factors that cause high latency and poor performance for groups of users. HIP aggregates transactions that have high latency due to same causes, and suggests values for causal variables that will alleviate the poor performance. HIP finds both immediate and indirect causes of high latency. Realizing HIP requires solving four challenges, which we describe below.

First, the search for alternate configurations must be fast and scalable. Because many factors affect latency, and these factors can take many values, a brute force or unintelligent search for alternate configuration will not scale. HIP addresses this problem by investigating what is causing high latency events in the CDN in the first place. In many cases, a few causes affect a large number of transactions, making it feasible to fix the problems with few changes.

Second, finding the underlying causes of high-latency transactions is difficult, for two reasons. (1) Many factors contribute to latency and a high-latency event can occur because of combination of one or more of these factors. For a given high-latency transaction, it is difficult to estimate which factor caused it. (2) Even when a model for predicting system response time based on causes is available, (e.g., using WISE [54]), it is difficult to map an observed response time to causes because the response-time function is typically not injective; that is, many transactions with similar response times may have different contributing causes. To solve this problem, HIP uses WISE to determine the causal factors and their interdependences. HIP then identifies factors that may be responsible for the high latency transactions by comparing values of causal variables for transactions with poor latency with transactions with good latency. HIP also groups transactions that have the same cause for high latency, allowing operators to focus on subsets of clients at a time.

Third, the appropriate corrective action may differ depending on what is causing the high-latency transactions. Many factors that contribute to latency are stochastic, which means that high-latency response time for a transaction could be because of a systemic problem in a contributing factor; on the other hand, the factor may simply be an outlier that coincides with the particular transaction. Whether the cause is systemic or coincidental can affect what corrective action an operator might take. For example, if a systemically large network round-trip time is causing low response times for clients, then the solution might be to fix routing or to deploy a server nearer to the clients. If, on the other hand, high latency is due to spikes in round-trip time, a possible solution might be to increase capacity. To distinguish coincidental causes from systemic ones, HIP analyzes the temporal behavior of the factor.

Finally, mitigating high-latency transactions requires not only identifying the responsible factors, but also proposing actions that can mitigate high latency. HIP produces scenarios with proposed changes to values of factors that will mitigate high-latency causes. CDN operators can evaluate these scenarios using WISE.

We have implemented HIP using a combination of Sawzall logs processing language [46] and R programming language [47]. We have used HIP to determine ways to improve network-level and browser-level latency for Web search service of a large provider (that we refer to as "Google" for anonymity). HIP determines that the factors causing high latency vary by region, but in most regions, fewer than three causes cover nearly 90% of the high-latency cases. In the USA, 84% of high-latency cases occur due to server-side contribution, but they are mostly of coincidental nature. Large network round trip time causes around 37% of high-latency cases: 2/3rd of these are coincidental, but other 1/3rd is systemic, happening under high load conditions. Around 19% of high-latency transactions are due to non-optimal frontend-backend pairings due to misconfiguration or overload. In Australia and India, client bandwidth and location of servers causes of more than 50% of high-latency cases. We find many cases of systemic causes where clients in certain ASes consistently experience poor performance[1].

[1]Percentages of HLTs affected by each factors add up to more than 100% because an HLT may occur due to one or more factors.

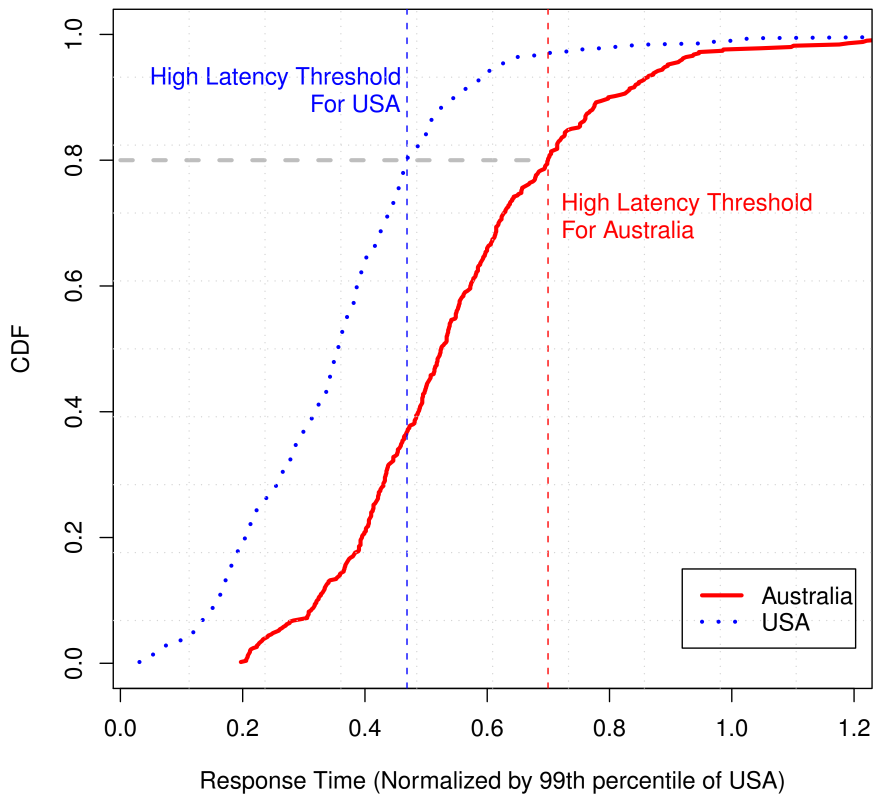


Figure 15: Example *how-to* questions. How to improve the response time for requests above the 80^{th} percentile for Australia or USA? How to make response time for users in Australia similar to that for users in USA?

This chapter is organized as follows. Section 3.2 describes problem scope and motivation. Section 3.3 presents an overview of HIP. It provides background on WISE, shows how it relates to HIP, and states intuition for various steps in HIP. Section 3.4 presents algorithmic details of HIP and Section 3.5 presents the implementation of HIP. Section 3.6 describes the CDN on which we evaluate HIP; we present the results of this evaluation in Section 3.7 by applying the technique to Web requests from a Web search service of a large provider. Section 3.8 discusses limitations and on going work for HIP.

3.2 Problem Context and Scope

In this section, we use the performance distribution of Google's Web search service for clients in Australia and USA to illustrate example *how-to* questions that may arise.

We explain a typical CDN setup and the potential actions in this CDN that HIP can suggest as part of answer to the *how-to* questions.

Example "How-to" Questions. Performance rankings among Web service providers is often based on mean response time or response time for high percentiles. Organizations also use these metrics as basis for their performance goals. Unfortunately, these metrics are sensitive to tail of the performance distribution. Figure 15 presents the CDF of network response time (*nrt*) measure for Web Search requests from Australia and USA for Google. We choose the 80% latency point as a cut-off for defining high-latency events. To perform well on the performance metric for Australia, service Google needs to address the following questions:

- **Question# 1.** How to reduce the cases of high-latency response time in Australia?
- **Question# 2.** How to make the response time for users in Australia comparable to response time for users in the USA?

HIP's goal is to answer questions of this nature. As we shall see shortly, HIP answers *how-to* questions of this nature by first finding the factors that are unique about the high-latency transactions. Specifically, HIP tries to find factors that can be mapped to actions that a CDN operator can take to achieve the service performance goals. In Figure 15, we define transactions with response time larger than the 80^{th} percentile as high-latency transactions. This threshold is in line with practice of using high percentiles to classify undesirable cases of latency.

Content Distribution Networks. Most CDNs use a two-tier architecture, like in Figure 16. The first tier is globally distributed front-end (FE) proxy servers that provide caching, content assembly, request pipelining, backend load balancing. The second tier comprises backend (BE) servers that implement the application logic, and which might also be replicated and globally distributed. The FE and BE servers may belong to a single administrative entity (*e.g.*, Google [7]), or to different administrative

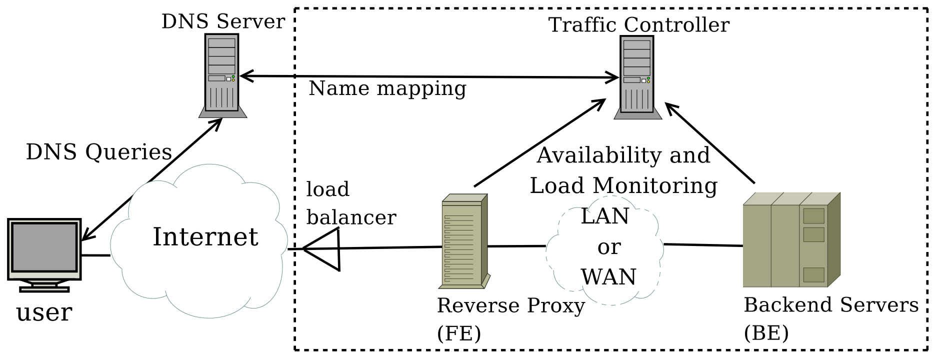


Figure 16: Typical CDN Architecture.

entities in case of commercial content providers, such as Akamai [2]. The network path between the FE and BE servers may be over a public network or a private (leased) network, or a LAN when the two are colocated in same data center.

CDNs typically use DNS redirection or URL-rewriting [6] to direct the users to the appropriate FE and BE servers. This redirection may be based on the user's proximity, geography, availability, and relative server load. To improve the network path between the users and FE servers, operators typically place FE servers in managed hosting locations, or ISP points of presence, that are closer to the eventual users. In some cases, FE is logically inside the CDN's IP network and the CDN buys transit or peers directly with ISPs to reach its users.

When a service provider or CDN operator wishes to improve service performance, it can take several types of actions depending on the cause of the poor performance. On the network side, it can deploy new frontends, new backends or parts of backend services closer to the user, it can decide to peer or buy transit from ISPs. On the content side, it can reduce content size or use different content encoding, or try to serve more content from the caches. It can try to increase backend or network capacity. The goal of HIP is to help operators decide which of these actions are more relevant for specific populations of users.

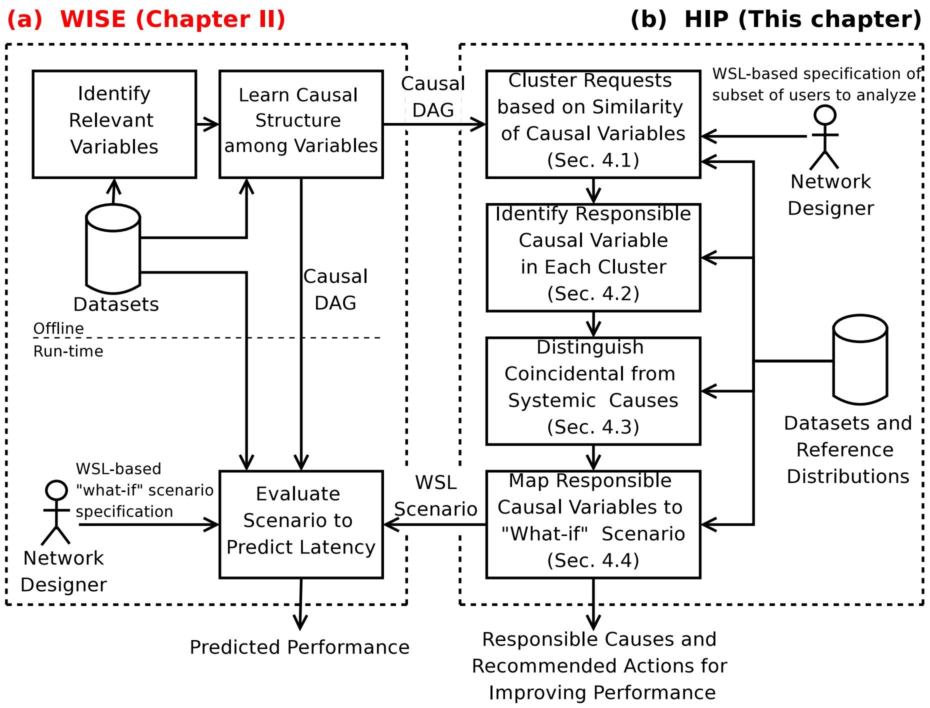

(a) WISE (Chapter II)
(b) HIP (This chapter)
Identify Relevant Variables
Learn Causal Structure among Variables
Causal DAG
Datasets
Offline
Run-time
Causal DAG
WSL-based "what-if" scenario specification
Network Designer
Evaluate Scenario to Predict Latency
Predicted Performance
WSL Scenario
Cluster Requests based on Similarity of Causal Variables (Sec. 4.1)
WSL-based specification of subset of users to analyze
Network Designer
Identify Responsible Causal Variable in Each Cluster (Sec. 4.2)
Distinguish Coincidental from Systemic Causes (Sec. 4.3)
Datasets and Reference Distributions
Map Responsible Causal Variables to "What-if" Scenario (Sec. 4.4)
Responsible Causes and Recommended Actions for Improving Performance

Figure 17: HIP approach and relationship with WISE.

3.3 HIP Approach

To answer *how-to* questions for performance improvement, HIP first finds the factors that are responsible for poor performance. HIP proposes fixing the values for these factors as answer to *how-to* questions. Figure 17 presents an overview of the HIPap-proach. To identify the causes HIP uses the causal dependency structure among latency causing factors in the CDN, which is generated using WISE [54]. Once HIP has identified the responsible factors, it recommends actions and produces *what-if* scenarios that an operator can evaluate using WISE. In Section 3.3.1, we provide a brief background on WISE. Section 3.3.2 summarizes the key steps in HIP approach and where applicable, describes how they relate to WISE. Section 3.4 will describe the specific algorithms in more detail.

3.3.1 WISE Background

WISE allows operators to specify scenarios with hypothetical or "*what-if*" values for one or more factors for a subset of clients in the CDN. WISE then uses the causal structure and conditional distributions expressed in the dataset to find the distribution of response time under the *what-if* values specified in the scenario. Figure 17(a) summarizes the high-level steps in WISE: (1) identifying variables in the dataset that affect response time; (2) learning the dependencies among the variables; and (3) estimating the response-time for the specified *what-if* scenario. WISE also includes a scenario specification language to facilitate the network designers. In the following we briefly describe these steps and refer the reader to WISE publication [54] for details.

Identifying Relevant Variables. The main input to WISE is a large dataset that represents the distribution of variables under a variety of conditions. As all causal variables raise probabilities of outcome [15, pp. 403-418] WISE retains the variables in this dataset that are correlated with service response time.

The set of factors in the dataset limits the nature of *what-if* or *how-to* questions that can be answered using that dataset. Therefore, choice of appropriate dataset as input is crucial to both HIP and WISE. We rely on a domain expert to provide this dataset.

Facilitating Scenario Specification. WISE includes an easy-to-use interface based on WISE-Scenario Language (WSL). WSL provides a set of built-in operators using which network designers can specify *what-if* scenario as relative changes to the values of variables in existing deployment or as new values from scratch for an arbitrary subset of clients.

To allow integration with WISE, HIP also uses WSL as interface with network designers. The designer specifies the subset of transactions or users that she wishes to analyze with HIP using WSL, and HIP produces the answers to the *how-to* questions in form of configuration changes expressed in WSL, which can be evaluated using WISE to estimate their effect.

Learning the Causal Structure in the Dataset. WISE learns the dependency structure among the variables and represents it as a *Causal Bayesian Network (CBN)* [45]. A CBN is a Bayes' Network—compact representation of conditional dependencies among variables—but the edges in the network have causal interpretation. It is represented as a Directed Acyclic Graph (DAG) in which nodes represent the variables and an edge from node x_i to x_j mean that change in x_i "causes" a change in x_j. CBN DAG satisfies the Markov Property: each variable, when conditioned on its parents, is independent of all other variables except its descendants. In an optimal DAG, no smaller set of parents satisfies the Markov property for any node.

As an example, consider a dataset with five input variables ($x_1 \dots x_5$), target variable y, and a CBN DAG in the Figure 3. If we change the value of variable x_2, then the distributions for variables x_1 and x_5 remains unchanged, but the distribution of the descendants of x_2 must change to maintain consistency with dependencies in the system.

Evaluating *what-if* Scenario. WISE uses the DAG to evaluate *what-if* scenarios. When the designer specifies a scenario with changes to a variable, WISE updates the value of all the descendants of the variable that is changed in the scenario. Because of Markov property, new value of the descendants depends only on the parents and not other variables. WISE uses a nearest-neighbor search to estimate the new values of affected variables.

3.3.2 HIP Overview

Figure 17(b) summarizes the HIP approach. Network designers use WSL to specify the subset of clients for which they wish to answer *how-to* questions with HIP. HIP then investigates causes of high-latency to determine actions that an help mitigate high latency. For this, HIP (1) Clusters high-latency transactions based on similarity of values of latency causing factors; (2) Finds variables responsible for high-latency in each cluster; (3) Distinguishes systemic causes from coincidental ones and (4) Maps causes to actions that can mitigate high-latency. In the following we provide a brief overview and intuition for these steps. Section 3.4 discusses the algorithmic details for these steps.

Using Causal DAG and Clustering on Causal Variables. The causal DAG that WISE produces is useful for answering *how-to* questions. In Figure 3, if the outcome variable (y) has an abnormal or undesirable value, then it must have been *caused* by abnormal value one or more of the variables that are immediate parents of target variable y in the DAG: *i.e.*, variables x_3, x_4 or x_5. We refer the immediate parent variables of the response time variable as proximate causes, and any ancestors of the proximate cause as high-level causes.

If the network designer can identify the variable(s) that caused high latency for specific transactions then performance can be improved by *fixing* this causal variable. Based on this intuition, HIP uses WISE to generate causal DAG for response time in the CDN. Then for high-latency transactions, HIP analyzes the values of proximate and high-level causes for abnormal values. Instead of finding the responsible cause one transaction at a time, HIP performs this operation in batches for similar transactions. This is discussed below.

Transactions that have high latency due to similar underlying causes should have similar value of the causal variables, and require similar solutions for fixing the latency issues. Identifying such groups can greatly facilitate the network designers because they can address a group of transactions with one or few actions. HIP finds such group by using clustering algorithms on similarity of causal variables among transactions. Section 3.4.1 describes the algorithmic details for clustering.

Identifying Responsible Variable for Each Cluster. To isolate the variable that is responsible for high latency, HIP compares the values of variables in each cluster values for transactions with normal latency. If the values are significantly different, then HIP considers this variable as the responsible variable. HIP relies on the operator to specify subset of transactions which have normal latency and are used as a reference for identifying responsible variables. Section 3.4.2 discusses this in more details.

Distinguish Coincidental and Systemic Causes. An inadvertent side effect of clustering transactions based on similarity of causes is that some clusters may be formed by grouping of coincidental outlying values of variables. Such coincidental outlying values are common in networks because of stochastic nature of queuing or processing times. From diagnosis and planning perspective, the solutions for *fixing* coincidental causes may be very different from solutions for fixing systemic ones. HIP uses temporal properties of variable values to separate coincidental and systemic causes. Section 3.4.3 presents details.

Mapping Responsible Variables to a *what-if* Scenario. HIP uses simple heuristics to map the responsible variables to a *what-if* scenarios that propose *fixed* values of causal variables as *what-if* values. HIP uses WSL to codify the *what-if* scenarios, which operators can readily evaluate using WISE. Section 3.4.4 provides more details of this mapping.

3.4 HIP System

This section describes the details of HIP system. Section 3.4.1 describes the rationale, challenges and HIP's algorithms for clustering high-latency transactions (HLTs). Section 3.4.2 describes the techniques for finding causal factors that are responsible for high-latency in each cluster of transactions. Section 3.4.3 describes the problem of coincidental causes and the method that HIP uses to distinguish coincidental causes from systemic causes. Section 3.4.4 describes how HIP maps the causes of high-latency to configuration change recommendations that network designers can make to mitigate HLTs.

3.4.1 Clustering High Latency Transactions

HIP uses clustering to find groups of HLTs that have similar causes. We first describe the rationale for clustering (§ 3.4.1.1) and then the algorithmic details of clustering algorithms that HIP uses (§ 3.4.1.2).

3.4.1.1 Rationale for Clustering

Clustering serves three purposes: (1) Facilitating operators by finding aggregation of transactions whose latency can be mitigated with similar actions; (2) Finding high-level causes, *e.g.*, whether the causal factors are affecting transactions from particular regions, or particular servers. This insight can be useful for the network operators; (3) Efficiency in finding responsible causal factors. We elaborate on these in the following.

Aggregation of transaction requiring similar actions. Clustering HLTs based on similarity of values of causal variables produces groups of HLTs that have high response time value due to similar reasons. As a result, it is likely that the latency of these transactions can be *fixed* by similar actions. Further, partitioning HLTs in clusters such that HLTs have high intra-cluster similarity and low inter-cluster similarity, then we can minimize the number of groups, each with a potentially different cause, and latency-mitigating solution. This is helpful for CDN operators because they wish to be able to *fix* large number of high-latency cases with few changes to the network. In Section 3.7.1 we show that HIP successfully produces clusters of transactions with high intra-cluster and low inter-cluster similarity.

Finding High-Level Causes using Recursive Clustering In the causal DAG (§ 3.3.2) the immediate parents of the response time variable are called proximate causes and the ancestors of proximate causes represent the high-level causes, or the factors that cause the values of proximate causes. For example, while network round-trip time (*rtt*), size of response (*sB*), frontend-backend network round-trip (*febe_rtt*), backend server processing time, may be proximate causes that contribute directly to transaction's response time latency, these proximate causes are themselves caused by higher-level factors such as location of users and servers, or frontend-backend pairings, or type of content that the user requests during a Web transaction.

While knowing the proximate cause of HLTs is useful, the causal explanation is attenuated and limited to proximate causes only. To make the inference more useful, we wish to map the proximate causes to higher-level causes which can further identify the specific subsets of transactions, geographical regions, servers, or other control factors that need to be modified to reconfigure the network and mitigate HLTs. As an example, if network *rtt* is primarily responsible for high-latency for a group of transactions, then the CDN operator would be interested in finding out whether the *rtt* is high for a particular subset of clients, such as from an AS, or /24 block, or those going to a particular server.

To find high-level causes, HIP clusters the HLTs that are already clustered based on proximate causes, using similarity on parents of proximate causes, and repeats the process recursively until it reaches variables that easily identify action. Together with information about proximate causes and the extent to which they have abnormal values, HIP is able to produce narrowed down and specific configuration scenarios that can mitigate HLTs.

Note also that if we cluster only on high-level causes, *e.g.*, AS number of the clients, then we can only infer an incomplete causal explanation: we might be able to say that a particular AS has many HLTs. Using hierarchical clustering produces more descriptive explanations, *e.g.*, an AS has high *rtt* which results in HLTs.

Efficiency in finding responsible causal factors. HIP isolates the factors responsible for high-latency by comparing the values of causal variables for HLTs with values of causal variables for normal latency transactions (§ 3.4.2). Because HIP clusters HLTs with similar values of causal variables, all the transactions in a cluster must have same responsible cause(s). Therefore, instead of finding responsible causal factors for individual HLTs, HIP can perform the comparisons with normal transactions at a group level, and find responsible causal factors for the whole group of HLTs with one comparison; this makes identifying responsible causal factors very efficient.

3.4.1.2 Clustering Algorithms

In this section we describe clustering algorithms that HIP uses to group HLTs with similar causes. We first describe the function for determining similarity; this is challenging because the factors are of mixed type and their distributions sometimes may have long tails, making them difficult to cluster. We then describe the clustering algorithm and how HIP finds the appropriate number of partitions that HLTs should be divided in to. Finally, because HIP uses recursive clustering to find out high-level causes, but recursive partitioning of data can produce partitions that have too few samples to make a statistically valid inference, we discuss how HIP ensures that there is enough data points (transactions) in each partition for at each stage of clustering.

Similarity Function. Latency-causing variables have mixed types (binary, continuous, ordinal, etc), different units and varied, and some times long-tailed distributions. This makes it challenging to combine these variables in a single similarity function. Following describes how HIP overcomes these challenges.

Long-tailed distributions and different units. Some latency-causing variables, *e.g.*, network round-trip time (*rtt*), may have long-tailed distributions. Samples in tail of the distribution can have large dissimilarity if computed on linear scale, making them hard to cluster. Additionally, variables may have different units and scale, for example, *rtt* is in units of time, but size of response from the server is in Bytes. Without careful choice of similarity function, values of one variable may overshadow other variables, making it difficult to assess similarity.

To overcome long-tailed distributions problem, HIP standardizes the logarithmic values of continuous or ordinal variables with long tails. i^{th} feature of j^{th} sample, x_{ij} in the dataset becomes:

$$\hat{x}_{ij} = \frac{\log(x_{ij}) - mean(\log(x_{*j}))}{sd(\log(x_{*j}))} \tag{1}$$

HIP standardizes continuous and ordinal variables that do not have long tails using a linear scale instead of logarithmic scale. Standardization also helps with problems of varying units and scales as similarity in on standardized values is somewhat scale-invariant.

Mixed data types. Some of the variables that affect response time do not have numeric values, for example the identifiers for servers may have nominal values. Variables such as the number of retransmits during a transaction, may have ordinal values. HIP uses Gower Similarity Coefficient [29] as similarity between samples of mixed data types. Gower Coefficient defines similarity of continuous or ordinal variables as: $s_{ijk} = 1 - \frac{|x_{ik} - x_{jk}|}{r_k}$, where r_k is the range (max - min) of the k^{th} variable. For binary and nominal variables, s_{ijk} is 1 if the values of x_{ik} and x_{jk} match, and 0 otherwise. Thus similarity on all dimensions is in interval $[0, 1]$. Similarity between two transactions i and j, s_{ij}, is sum similarity on all the factors. We can convert similarity coefficient into dissimilarity as $N - s_{ij}$, where N is the number of factors on which clustering is performed.

Clustering Algorithm. HIP uses Partitioning Around Medoids (`pam`), a common implementation of k-medoid clustering. `pam` is more robust to noise than k-means, although `pam`'s computational complexity is somewhat higher. Spectral clustering algorithms, such as, Eigen Cluster [13], are more robust than `pam`, but they are computationally much more expensive, making them unsuitable for HIP which clusters repeatedly for each scenario it analyzes.

`pam` is sensitive to choice of k, the number of clusters. HIP uses Gap statistic [24, pp. 519] to determine optimal k. This heuristic estimates Gap statistic for increasing values of k, until the within cluster dispersal flattens. HIP finds optimal numbers of clusters separately for each dataset that it clusters.

Scoring To Increase Samples in Clusters. To manage the computational overhead of clustering, HIP uses a small-sized samples (typically comprising few thousand transactions) to perform first stage clustering. The number of transaction samples in each cluster after partitioning is typically only 5-10% of the size of the original sample, comprising only few hundred transactions. Performing recursive clustering on such small datasets can lead to erroneous and noisy inference. HIP uses *scoring* to increase the size of samples in a cluster. HIP assigns a transaction i in the dataset to a cluster C_k, whose medoid is m_k is most similar to transaction x_i, while ensuring that similarity of x_i with medoid of cluster k is greater than the average similarity (S_k) of the transactions in the cluster with the its medoid.

$$C_i = \underset{\substack{k=1 \ldots K \\ \text{Gower}(x_i, m_k) > S_k}}{\arg\max} \quad \text{Gower}(x_i, m_k) \tag{2}$$

HIP assigns transactions to clusters until each of the K clusters have a certain minimum number of samples. We use $\min(2000, 100 \text{ x } k)$ as the minimum threshold for number of transactions that we need in a dataset before we proceed with partitioning it in k clusters. These thresholds are driven by limitations of our implementation (§ 3.5). HIP uses scoring for second and higher stage clustering, as well as for distinguishing coincidental and systemic causes (§ 3.4.3).

3.4.2 Identifying Responsible Causal Variables

Clustering produces groups of transactions that have similar values of causal factors for latency, but does not provide information about which of the factors are causing higher latency. To address this, HIP compares the average value of each of the factors in a cluster with the value of causal variables for transactions that have normal latency. HIP uses reference distribution to obtain transactions that have normal latency. For example in cases presented in Figure 15, head of the latency distribution for Australia, or a latency distribution of another region, USA can serve as the reference distribution. The causal variables for transactions in a cluster whose value is *abnormal* compared to values in the reference distribution, are considered responsible for high latency. The intuition is based on exclusion of other potential factors as causes of high latency. If the values of all the other variables is approximately same as normal value, those variables could not be responsible for higher latency.

HIP uses a simple thresholding based-heuristic to determine whether a variable has abnormal value in a cluster: variable x_i is responsible factor in cluster k if $mean(x_i^k) \geq D_i^{-1}(u_{upper})$. Here $mean(x_i^k)$ is the average value of variable x_i for transactions in cluster k, and $D_i^{-1}(u)$ denote the u^{th} percentile of variable i from the reference distribution. If x_i is negatively correlated with response time (*e.g.*, *client bandwidth* is often negatively correlated with response time) then we test for $mean(x_i^k) \leq D_i^{-1}(u_{lower})$. u_{upper} and u_{lower} are tunable percentile thresholds for deciding positively and negatively correlated abnormal variables, respectively. We use 80% and 20% as values for these thresholds respectively in our implementation to align them roughly with the high-latency threshold.

3.4.3 Distinguishing Coincidental & Systemic Causes

Many factors that contribute to latency are stochastic, which means that high-latency response time for a transaction could be because of systemic problem in a contributing factor or simply an outlier value of one of the contributing factor that coincides with the particular transaction.

Distinguishing the two types of causes is important because the actions needed to mitigate the causes might be different depending on the type. If a responsible causal variable is coincidental, then CDN operator might need to focus on reducing the

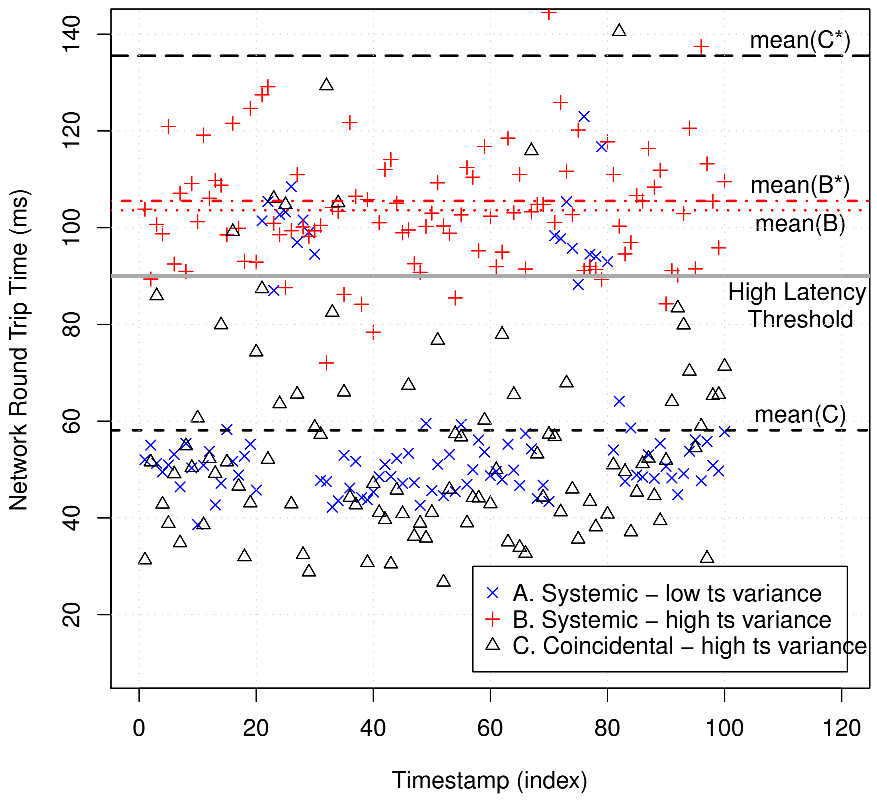


Figure 18: Timestamp behavior for systemic and coincidental causes. Network RTT is simulated.

common-case variance of the variable. For example, if backend server time *be_time* is deemed a coincidental cause, then CDN operator might consider increasing capacity, or making algorithmic changes in the server logic that limits the maximum delays in responses from the servers. If responsible variable is systemic, such as in case of clients for whom *rtt* is consistently high, then depending on the nature of the cause, the CDN operator may need to make configuration changes, such as mapping users to different server, or adding a new peering connection, or deploying a new datacenter.

Distinguishing Coincidental and Systemic Causes. Unfortunately, clustering merely on similarity of values of causal factors may groups transactions with coincidental and systemic causes together. To distinguish the two types, when HIP detects a variable as responsible, it considers three cases.

A. Variable is a systemic cause but it occurs in only at certain periods of time, such as, under high load.

B. Variable is a systemic cause that occurs all the time.

C. Variable is a coincidental cause that occurs occasionally due to inherent variability of stochastic process.

Figure 18 illustrates these cases with a simplified example. Each point on the figure represents *rtt* variable for a transaction. The transactions above the high-latency threshold (HLT) are above the marked line and are grouped in a cluster, though they may belong to different cases. Type A HLTs have high latency because *rtt* is high for certain periods (timestamp intervals 20–40 and 70–90). Type B transactions have a consistently large value of *rtt*. Type C transactions have an occasional large value of *rtt*. The horizontal lines labeled mean(X) mark the mean *rtt* for all transactions of type X, and lines labeled mean(B*) and mean(C*) mark the mean *rtt* for HLTs of type B and C respectively. We do not show mean(A) or mean(A*) to avoid clutter and because HIP does not rely on these for identifying type-A HLTs.

The three cases of transactions have a unique combination of variance of their timestamp and differences of mean values of *rtt* for HLTs and normal latency transactions. Timestamps for HLTs in case A are confined to certain periods, resulting in low variance on timestamp for high-transaction cases. Timestamps for cases B and C, both have high variance on timestamp. However, mean(B*)-mean(B) is small, but mean(C*)-mean(C) is large. The intuition behind for this is that if an variable is a coincidental cause, only abnormal values of that variable will occur in the cluster, and bulk of the distribution will be outside the cluster, causing the overall mean to be significantly different than the mean in the clusters. HIP uses these observations to distinguish between the three cases. HIP distinguishes between systemic and coincidental causes based on these observations.

The mean of responsible variable in clustered transactions only represents mean(X*). To find mean(X), HIP has to find other transactions that are similar to the ones in the cluster, but for whom the responsible causal variable may have normal value. For this, HIP uses scoring to find transactions from the dataset of the head of the distribution that are similar to the medoid of the current cluster on the value of all the variables, ignoring the value of the responsible variable. These transactions represent the transactions that are similar to the HLTs in the cluster except for the value of the responsible cause, but not necessarily have high latency: referring to Figure 18, these transactions may be both above or below the high-latency threshold. Once sufficient number of such transactions are found, HIP computes mean(X) as the mean of value of responsible variable for the newly found transactions, and then compares it with the mean(X*) to distinguish cases B and C.

3.4.4 Mapping Causes to Remedies

HIP uses a simple heuristics to map identified causes to a *what-if* scenario representing actions that may remedy HLTs. After performing recursive clustering and grouping HLTs based on both proximate and high-level causes, HIP produces a *what-if* scenario with new values of responsible causal variables for the group of HLTs. HIP simply proposes the average values of the responsible proximate cause in the reference distribution as the new values. HIP scenario using WSL so that it can be readily evaluated using WISE. Presently, HIP only produces scenarios for systemic causes. For coincidental causes, it generates a report explaining the factor.

The intuition behind the heuristic for mapping causes to remedies is simple: if a factor caused HLTs, then setting its value to a value for normal latency transactions should reduce the latency for the HLTs in question, bringing it in line with the latency for normal transactions. However, this approach is naive in many ways, such as, it does not consider whether an action is feasible, or whether changing one causal factor may inadvertently change other factors, or that there may be other ways besides fixing the identified cause for mitigating latency. We discuss these limitations in Section 3.8.

We are working on alleviate these limitations in HIP.

3.5 Implementation

We have implemented HIP using a combination of Sawzall logs processing language [46] and R programming language [47]. We use Sawzall to collate datasets that we obtain from FE and BE servers in the CDN, and also use Sawzall to obtain datasets that represent high-latency transactions and the reference distributions that HIP user is interested in analyzing. In our current implementation, all the remaining processing is done using R. We have chosen R because it has a wealth of libraries for statistical inference and allows rapid prototyping. We have also implemented a prototype for WISE using R that we use to evaluate the configuration changes that HIP suggests as part of answering *how-to* questions (§ 3.7.4). We have also implemented an simplified version of WISE [54] using R to estimate the effect of actions that HIP produces.

While R allows rapid prototyping, the implementation is not scalable, and puts severe restrictions on the size of datasets that HIP can cluster: clustering 10,000 transactions exhausted 2GB memory on our desktops. Our current implementation uses scoring (cf. § 3.4.1.2 to work around this problem. We are working on porting our R-based implementation to Map Reduce [16] framework.

3.6 Evaluation Environment

We have used HIP to evaluate causes of high-latency transactions (HLTs) for Google's Web search service. Google's Web search service CDN architecture is similar to CDN in figure 16, comprising globally distributed frontend(FE) servers that receive user's query and forward it to search backend (BE) clusters. In the following we describe the Web search transaction and how we measure its response time (§ 3.6.1). We describe the dataset from Google's CDN and causal DAG obtained from this dataset using WISE (§ 3.6.2). We have used HIP to answer *how-to* questions for mitigating HLTs of Web search service in three countries: USA, Australia and India. In Section 3.6.3

Table 4: Variables related to *nrt* in the dataset from Google's CDN. Variable Types: Binary (B), Continuous (C), Nominal (N), Ordinal (O).

Variable	**Type**	**Description**
ts, tod	*C*	A timestamp of instance of arrival of the request at the FE. All FE use same time zone. Hourly and second-level time-of-day (tod) is derived from the timestamp.
srP	*O*	Number of packets retransmitted by the server to the client, either due to loss, reordering, or timeouts at the server.
sB	*C*	Size in bytes of the encoded response sent by the FE server.
region	*N*	One of following: user's IP address, /16, /24 network prefixes, AS number, or geographical mapping to state and country.
fe, be	*N*	Identifiers for the FE data center at which the request was received and the BE data center that served the request.
rtt	*C*	Round-trip time between the user and FE estimated from the initial TCP three-way handshake.
bw	*C*	An estimate of access network bandwidth for the /24 IP address block for the user's IP address. It is estimated as the 90% TCP throughput for unthrottled responses greater than 4KB sent from FE to the IP addresses in /24 prefix.
febe_rtt	*C*	The network level round-trip time between the frontend and the backend clusters.
be_time	*C*	Time taken by BE to process the request forwarded by FE.
nrt	*C*	Network-level response time for the transaction as seen by the FE (see § 3.6.1).

we describe the CDN and user's access network properties in these countries, and demonstrate that they present a suitable environment to evaluate HIP. In Section 3.7 we present the results for applying HIP to answer *how-to* questions for these countries.

3.6.1 The Web Search Transaction

In this section we first describe the messages and events during a Web search transaction. We then describe the latency metrics and factors that contribute to latency.

Messages and events in the transaction. A Web search transaction comprises a user sending an HTTP request containing a query to Google, and Google's servers responding with results using an HTTP response. Figure 20 shows the messages exchanged during a Web search transaction with Google. When a user issues a Web search request to Google, Google's DNS-based request redirection system redirects the user's queries to one of the FE in the CDN. The FE process forwards the queries to the BE servers, which generate dynamic content based on the query. The FE caches static portions of typical reply, and starts transmitting that part to the requesting user as it waits for reply from the BE. Once the BE replies, the dynamic content is also transmitted to the user. The FE servers may or may not be collocated in the same data center with the BE servers. If they are collocated, they can be considered to be on the same local area network and the round-trip latency between them (*febe_rtt*) is only a few milliseconds. Otherwise, the connectivity between the FE and the BE is typically on a well-provisioned connection on the public Internet. In this case the latency between the FE and BE can be several hundred milliseconds.

Transaction latency. We measure the latency of Web search transaction as network-level response time (*nrt*) and browser-level response time (*brt*). In this paper we focus on answering *how-to* questions based on *nrt* alone. a transaction is the time between the instance when the user issues the HTTP request and the instance when the last byte of the response is received by the users. We estimate *nrt* from the server end as the sum of the round-trip time (*rtt*) estimate obtained from the TCP three-way handshake, and the time between the instance when the request is received at the FE and when the last byte of the response is sent by the FE to user. Key contributors to *nrt* are: (i) the transfer latency of the request from the user to the FE, (ii) the transfer latency of request to the BE and the transfer latency of sending the response from the BE to the FE, (iii) processing time at the BE, (iv) TCP transfer latency of the response from the FE to the client; and (v) any latency induced by loss and retransmission of TCP segments. A high-latency Web search transaction can occur due to unusual contribution from one or more of these factors.

3.6.2 Dataset & Causal Dependencies

We use data from an existing network monitoring infrastructure in Google's CDN. FE and BE servers export reports with values for many performance-related variables. FE and BE servers add unique identifiers to data exported for each transaction. We use this identifier to collate the records in dataset obtained from FE and BE servers. Table 4 describes the features in the dataset that had positive or negative correlation, and thus potential causal relationship with network-level response time.

Causal dependencies among latency-causing factors We use a dataset collected in July 2009, comprising 50 million transactions from about 10,000 ASes, and estimate the causal structure in this dataset using WISE. Figure 21 presents this causal structure using a DAG.

This causal dependency structure has several features that can aid answering *how-to* questions with HIP. First, Tariq et al. [54, 53] have used a similar dataset and DAG to predict response time for Web transactions with high accuracy. Therefore, we believe that this DAG is complete and accurate: it captures the necessary dependencies so that for every high-latency transaction, the *latency should be explainable by values of one of the proximate causes of* nrt *in the DAG.*

Second, *the causes are easily mappable to actions.* Poor *rtt* may point to need for a new FE closer to the clients, or a new peering connection if there is already a datacenter that is near the affected clients. Too many retransmitted packets (*srP*) may point to need for tuning server parameters or increasing network capacity to the clients. Large *be_time* may indicate need to reduce server response time or increasing capacity. Large content size (*sB*) may point to need for limiting the size of response, such as by using compression or composing smaller Web pages.

Third, the proximate causes or high-level causes of *brt* (parents of proximate causes) *hint at the place in network where the configuration change needs to happen to*

Table 5: Average CDN and user's access network characteristics in USA, Australia and India. For confidentiality, values of $\overline{rtt}$ and $\overline{febe\_rtt}$ and $\overline{nrt}$ are normalized by means of corresponding factors in USA.

Country	Local FE	Local BE	$\overline{bw}$ (kb/s)	$\overline{rtt}$	$\overline{febe\_rtt}$	$\overline{be\_time}$	$\overline{nrt}$
USA	✓	✓	5900	1.0	1.0	1.0	1.0
Australia	✓	×	4300	2.2	7.8	0.91	1.56
India	✓	×	700	3.4	6.5	0.94	1.73

mitigate high-latency transactions: These features identify infrastructure components such as frontend (*fe*) and backend (*be*) servers, locations of clients (*region*), as well as the properties of content, e.g., size of the content (*sB*).

3.6.3 CDN and Access Network Environment

We use HIP to determine causes for HLTs from three countries: USA, Australia, and India. We choose these countries because each is different from the other in terms of extent of CDN infrastructure that Google has in that country, as well as in terms of the access network conditions for users. This allows us to evaluate HIP under a wide variety of real network conditions. Table 5 summarizes the characteristics of CDN and user's access network in these countries. USA is very well provisioned and has many FE that are close to the users, resulting in smaller *rtt*. Many FE servers are collocated with BE servers, resulting in smaller *febe_rtt*. Access networks for users are also well provisioned and client's network bandwidth is high. Australia does not have a BE in the country, but users have good access network conditions. India also does not have BE servers locally and the access network conditions are generally worse of all three countries. Average *nrt* for transactions in Australia and India is 1.5 times and 1.7 times the average *nrt* in the USA, respectively.

3.7 Evaluating HIP on a Real CDN

In this section, we use HIP to answer *how-to* questions about mitigating HLTs that we posed in Section 3.2. In Section 3.7.1 we present the clustering of HLTs and show

Table 6: Highlights of results in Section 3.7.

Summary of causes (§ 3.7.3, § 3.7.2, Table 7)
be_time is dominant cause of HLTs in USA and Australia.
rtt and *bw* are dominant causes of HLTs for India and Australia.
Non-optimal FE-BE pairings cause 19% of HLTs in USA.
Large response size is causes 13% of HLTs.
In Australia and India, large *rtt* and *febe_rtt* inhibit response time latency comparable to USA.
. **Examples of systemic causes (§ 3.7.2.2)**
AS-AUS2 is a mobile wireless network.
An FE on USA east coast using a BE on the west coast.
Examples of coincidental causes (§ 3.7.2.2)
High *be_time* is alway coincidental.
Many ASes in USA have HLT due to coincidental large *rtt*.
Example of mitigated high-latency transactions (§ 3.7.4)
Network response time for clients decreased by 11% after fixing the FE-BE pairing.

that HIP produces clusters of transactions with high similarity. In Section 3.7.2 we address how to mitigate HLTs in Australia, India and the USA. In Section 3.7.3, we explore what causes need addressing to make response time in Australia and India comparable to the response time in the USA. Finally, in Section 3.7.4, we demonstrate that the actions that HIP proposes can mitigate HLTs. Table 6 presents highlights of results in this section

3.7.1 Clustering High-latency Transactions

Figure 19 shows results for clustering HLTs from the three countries based on similarity of proximate causes of *nrt* variable, shown Figure 21. In each case, we use the 80^{th} percentile of *nrt* to classify a transaction as a HLT. The optimal number of clusters for USA, Australia, and India are 28, 24 and, 35, respectively. Each panel represents a sample of HLTs. These transactions are ordered based on their cluster affiliation, and clusters are ordered by their size. Point (i, j) on the graph presents a color encoding of dissimilarity between transaction i and j. Darker shades represent low dissimilarity and lighter shades represent high dissimilarity.

HIP clusters the transactions so that intra-cluster HLT dissimilarity is minimized and inter-cluster HLT dissimilarity is maximized. This is shown with strings of dark blue clusters along the diagonal in each panel in Figure 19. Recall that each factor used in the similarity function contributes a maximum dissimilarity of 1 unit (§ 3.4.1). Because *nrt* has six parents (Figure 21), the worse case dissimilarity between a pair of transactions can be 6. Average intra-cluster HLT dissimilarity less than 0.2 units in nearly all the clusters for all three countries. Average dissimilarity between HLTs in different clusters is about 3 units.

To further confirm the similarity, we separately test the variance of values of causal variables for HLTs within each cluster, and find that variance is almost always small for all the factors. Producing clusters with highly similar HLTs is important for identifying groups of transactions whose latency can be mitigated by similar actions. HIP succeeds in this goal.

A secondary goal in HIP is to minimize the number of groups of transactions which require separate actions. For this, HIP must produce clusters that are well-separated: *i.e.*, inter-cluster dissimilarity is high. In Figure 19 some boxes of darker shade that are not along the diagonal imply that transactions in two separate clusters also have low dissimilarity. This occurs because in some cases clustering fails to maximize inter-cluster dissimilarity, however few such cases do no not pose a problem in HIP because minimizing the number of clusters is only a secondary goal.

HIP also produces clusters with low intra-cluster and high inter-cluster dissimilarity for higher-stage clustering. The problem of occasional low inter-cluster dissimilarity that we observe in first stage clustering virtually disappears for higher stage clustering because there are fewer features that are parents of responsible causal variables, and in many cases, the features have nominal values with limited range, making it easier to produce well-separated clusters.

3.7.2 How to mitigate HLTs in Australia, India, and the USA?

In this section, we use HIP find causes of HLTs in Australia, India and the USA. We cluster the HLTs from each country as described in § 3.7.1. To identify the causal variables responsible for high latency, we use entire distribution of *nrt* for each country as reference distribution. We present summary of most important latency-causing factors (§ 3.7.2.1). We then provide a breakdown of HLTs by cause, and distinguish systemic causes from coincidental ones, and provide high-level groups of HLTs (§ 3.7.2.2).

3.7.2.1 Summary of HLT causes

HIP identifies responsible causal factors for each HLT cluster and produces remedies that can mitigate latency. Due to limited space, instead of presenting cause for each cluster, we summarize variables responsible for HLTs in each country: For each variable, we aggregate HLTs from all clusters in a country in which that variable is responsible for contributing to high latency, and present aggregate statistic in Table 7(a). We present the percentage of HLTs affected by the causal variable, as well as the mean and inter-quartile range of the causal variable for the HLTs in the clusters as well as for the reference distribution. This shows that value of causal variable for the HLTs was indeed undesirable compared to its value for normal transactions. For each country, we present the causal variable in order of the percentage of HLTs that the variable contributes to in the country. A transaction may be affected by multiple therefore the percentage of affected HLTs may not add up to 100%. We now summarize the most important causes.

Backend server processing time (*be_time*) is the dominant cause for high latency in the USA: 84% of all HLTs are caused by large *be_time*, almost twice (1.90) as much as the *be_time* for transactions with normal latency in the USA. *be_time* also contributes to 70% and 36% of HLTs and in Australia and India. The distribution of *be_time* is nearly identical in the three countries (cf. Table 5). The difference in the contribution is because the threshold for HLT is different in the three countries. In the USA, other factors already have acceptable values, so *be_time* stands out as a cause. In Australia and India, network-side problems overshadow the role of *be_time*.

Network round-trip time (*rtt*) is the dominant contributor for high latency in India, affecting 73% of HLTs there. *rtt* affects 37% and 47% of HLTs in the USA and Australia. In the USA, average *rtt* less than half of that in Australia or India (cf. Table 5), however there is significant temporal and regional variation within the country which contributes HLTs. India and Australia are also large countries, but Google does not have as many FE servers in those countries as it has in the USA, which causes generally higher *rtt*.

In Australia, HIP finds that the HLTs caused by large *rtt* are dominated by only 4 ASes; they account for 71% of such HLTs. One of the AS is served from FE in Australia, but is a mobile 3G service provider. Two of the ASes have persistently high *rtt* because they are served from FE in the USA. The fourth AS is served from FE in Australia, but some transactions coincide with high *rtt*.

In India, poor network conditions and low access network bandwidth results in large variance in *rtt* that then contributes to HLTs. We also found some cases of non-optimal mapping of clients to FE in India, which caused systemic problems. Poor *rtt* combined with low bandwidth also causes packet loss, as well as spurious and loss recovery retransmits of packets, which further aggravates *nrt*. HIP identifies *srP* as one of the significant contributing factors in Australia and India.

Frontend-backend round-trip time (*febe_rtt*) is generally low in the USA, but HIP identifies cases of misconfiguration or overload that result in FE pairing with far away BE for some transactions. *febe_rtt* affects 19% of HLTs in USA. It is not a HLT-causing factor in India or Australia because *febe_rtt* is also for normal latency transactions in those countries.

Size of the server response HIP identifies (sB) as a contributing cause for 13%–18% of HLTs in all three countries. We believe that it is driven by the user's choice of requesting many results for a search query. Such cases could be mitigated by limiting the size of the response or compressing the response before sending it to the user.

3.7.2.2 Distinguishing systemic and coincidental causes

In this section, we present a breakdown of causes of HLTs by high-level causes and analyze whether the causes are systemic or coincidental.

Backend server processing time (*be_time*). HIP found no systemic pattern for *be_time*: transactions from all the backend servers are almost equally likely to experience high latency, and we also did not find significant time-of-day effects either. We believe it is coincidental: the variance in the backend processing time is large and transactions coinciding with large processing times end up as HLTs.

Network round-trip time (*rtt*). In Australia, 70% of *rtt*-caused HLTs come from only four ASes; we refer to these as AS-AUS1, AS-AUS2, AS-AUS3 and AS-AUS4 for confidentiality reasons. These ASes contribute 26%, 23%, 13%, and 9% of HLTs, respectively. HIP found that almost 90% of transactions from AS-AUS1 and AS-AUS3 are being served from an FE on the west coast of the USA, resulting in very large *rtt*. *rtt* is a systemic cause for these ASes.

HIP found that although transactions from AS-AUS2 are served from a FE in Australia, almost all its transactions are HLTs, and they have a systemically high *rtt*; `whois` reveals that AS-AUS2 is for Hutchison Australia's 3G service. Cellular 3G is known to have high latency on the wireless hop, which may explain the high latency as a systemic cause.

AS-AUS4 uses the FE in Australia, but 14% of its transactions as HLT. The variance of hour-of-day variable for both normal transactions and HLTs is identical and high. HIP also did not find any subgroups of low-bandwidth clients among the HLTs. As a result, *rtt* is a coincidental cause for HLTs from AS-AUS4.

Clusters of HLTs in USA for which HIP determined *rtt* as responsible cause include transactions from 252 ASes. We describe results for most popular two. Lets refer to these as AS-USA1, AS-USA2. These ASes contributed 13% and 10% of the HLTs, respectively. For AS-USA1, 65% of HLTs are served at the primary FE, and all have high latency due to coincidental *rtt*. For the two secondary FEs, which served 21% of the HLTs from AS-USA1, *rtt* is Type A Systemic (ref § 3.4.3): HIP observed very low variance on hour-of-day variable. *rtt* between AS-USA1 and these FE is normal outside of hours when we observed high latency. For AS-USA2, all HLTs caused by *rtt* are coincidental.

Nearly two-thirds of HLTs in India for which *rtt* is responsible originate in one AS (AS-IND1). AS-IND1 is primarily served from an FE in India, but about 2% of AS-IND1 transactions are served from FE in Europe and Asia Pacific. For these transactions, *rtt* is a systemic and persistent cause, and they account for roughly 7% of HLTs from AS-IND1 that are caused by *rtt*. Unfortunately, we were not able to find a cause for why some these transactions are not served from FE in India. In the HLTs served in India, the average HIP shows that *rtt* is nearly three-folds the overall average for transactions from AS-IND1 served in India. There is only a slight dependency of *rtt* on hour-of-day, implying that these are either coincidental or there is a hidden systemic cause inside AS-IND1. HIP found that these HLTs cluster based on client access bandwidth *bw*, but found no explanation based on geographical or routed-prefix affiliation of clients who originated the transactions. Another 15% of *rtt*-caused HLTs originate in an AS that we refer to as AS-IND2. This AS is primarily served from a FE in Asia Pacific. Average *rtt* for HLTs from AS-IND2 is nearly twice the average for normal transactions. Like in the case of AS-IND1, although HIP is not able to find a further grouping based on region or server association that would explain large *rtt*, it finds that HLTs cluster based on access network bandwidth (*bw*). HIP declares *rtt* as systemic but high variance (Type B - cf. § 3.4.3) cause in both cases.

Frontend-backend round-trip time (*febe_rtt*). High-latency transactions caused by *febe_rtt* in USA are served from 8 different FE servers. We present details for top 2, which account for 72% of the HLTs caused by *febe_rtt* in the USA. About 40% of the USA traffic that the first of these FE served, went to its secondary backend for which the *febe_rtt* is twice that for its primary backend. All the transactions served from the secondary backend are HLTs which HIP detects successfully. Variance of the hour-of-day variable for the HLTs affected by this FE is low, (only 6.47), suggesting that Google uses the backend at only certain periods of time, perhaps for load reasons. The second most popular FE also uses the same BE using a similar pattern. *febe_rtt* is thus another Type A systemic cause.

We find that the causes that HIP identifies generally agree with intuition, given the CDN and access network characteristics in the three countries (cf. Table 5). HIP quantifies the causes and aggregates HLTs so that they can be fixed using few actions. HIP also identifies additional causes that are not immediately clear from high-level summarization of characteristics in those countries.

3.7.3 How to make latency comparable to the USA?

We use HIP to find the causal factors that make the response-time latency for transactions in Australia and India different from latency in the USA. To do so, we set the mean *nrt* in the USA as the high-latency threshold for transactions in Australia and India. This corresponds to the 30^{th} percentile and 39^{th} percentile of latency for Australia and India respectively. We then cluster the transactions with response time above this threshold (HLTs) using HIP. The optimal number of clusters for these HLTs is 32 and 41 respectively. We then use the latency distribution in the USA as the reference distribution and identify causes for each cluster of HLTs. Table 7(b) presents the mean and inter-quartile range for the factors that contribute to most HLT clusters. Third column in the table lists the total percentage of HLTs that are affected by the causal factor.

HIP identifies that *febe_rtt* and *rtt* are primarily responsible for inhibiting performance in Australia and India. Almost all the transactions in India and Australia with response time greater than the average in the USA have *febe_rtt* and *rtt* that is larger than the 80% in the USA. This is somewhat expected because neither country has a backend datacenter and the density of FE deployments in both countries is far less than USA as evidenced by higher average *rtt* (cf. Table 5). Not only does HIP identify the factors that capture this (somewhat obvious) reality, it also quantifies the differences compared to average transaction in the USA. In case of India, HIP also identifies that *bw* is an inhibiting factor, implying that even with improvement of CDN infrastructure may not achieve USA-like performance.

Also note that *be_time* does not appear as an inhibiting cause for making latency similar to that in the USA, although it is an important contributor for HLTs in the respective countries (see Table 7(a) and § 3.7.2.1). This is because the goal here is to make latency similar to that in the USA. HIP identifies that *be_time* is not an inhibiting factor because its distribution is nearly identical in the USA, India and Australia.

3.7.4 Evaluating Answers to How-to Questions

In Section 3.7.2.1 we showed that HIP finds two cases where an FE in USA is using a BE server which is situated far away for about 15% of its traffic, although a local, close-by BE is available. HIP identifies *febe_rtt* as a cause and proposes an action with new mapping of culprit FE to a nearby BE. We evaluate this configuration using our implementation of WISE. Figure 22 presents the current and estimated new distribution of *nrt* for all the clients of this FE, including those that are already served from optimal BE servers.. Mean latency for clients is reduced by and mean latency for Australia is reduced by 11% by just one action.

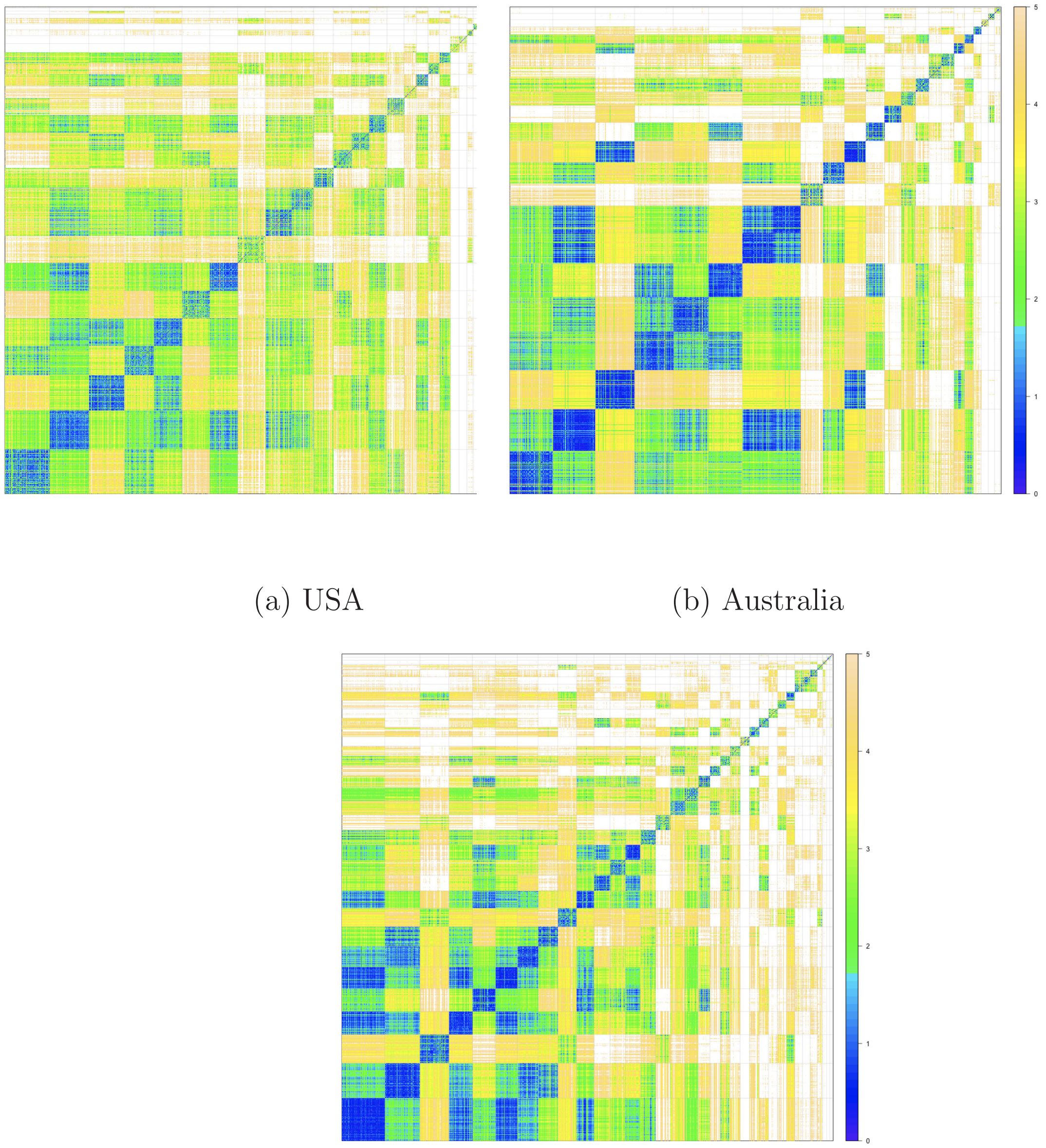
3.8 Discussion

In this section we discuss various limitations of HIP. Most of these relate to mapping causes to remedies.

Suggesting alternate causes and explicit remedies. Currently HIP does not provide explicit remedies. For example, HIP may suggest that the operator improve performance by reducing RTT for a subset of clients, but it does not make explicit, specific suggestions about how to reduce the RTT for those clients. Ideally, HIP might provide additional information about how to reduce RTT, such as increasing peering at various geographic locations. A new peering or transit relationship can affect BGP-level routing for many ASes, and could provide valuable information to the operator about how additional peering could affect overall response-time distributions. Unfortunately, specifying a scenario with correct value of network round-trip time (RTT) distribution for evaluating a potential peering or data center deployment has proven to be difficult, but existing tools such as WhyHigh [32] could complement HIP by providing such values to HIP's scenario evaluator.

Sometimes causes are not fixable. HIP assumes whatever is causing high latency for clients is fixable; unfortunately, however, this might not always be the case. For example, if transaction times are high due to high round-trip times, this underlying cause may be inherently fixable by adding additional peering, deploying new front-ends in data centers, etc. On the other hand, if a group of clients is seeing slow response times as a result of limited access bandwidth at the client, the CDN operator has very little control over direct fixes; in this case—and in general—the operator might to evaluate multiple possible remedy to a particular cause of high latency, in case the first remedy is either not possible or otherwise undesirable. For example, for cases where high latency is caused by low bandwidth or large *rtt*, it may be possible to mitigate HLTs by sending smaller responses. Currently, HIP does not suggest such alternate fixes, but we are working on extending HIP to provide this function.

Fixing one cause may inadvertently affect another cause. It is possible that a cause is fixable, but fixing this cause inadvertently changes the value of another latency-cause factor. Presently, HIP does not recognize such dependencies. Unfortunately, sometimes, these dependencies can undo the improvement in latency that may be achieved by fixing the cause that HIP had identified. As an example, consider the case of AS-AUS1 in Australia. It is being served from an FE in USA although there is an FE in Australia. However, the FE in USA is collocated with a BE to the *febe_rtt* is very small. There is no BE in Australia, therefore the *febe_rtt* for the FE in Australia is large. It is possible to fix the *rtt* for AS-AUS1 by serving it from FE in Australia, but possible improvement in response time due to reduced *rtt* may be somewhat offset by the increased *febe_rtt*. In Section 3.7.2.2, we show that HIP identifies *rtt* as a systemic cause because *rtt* is consistently large for AS-AUS1. However, fixing this cause may not fix latency.



(a) USA (b) Australia

(c) India

Figure 19: Size of clusters and dissimilarity among transactions after clustering for proximate causes of *nrt*.

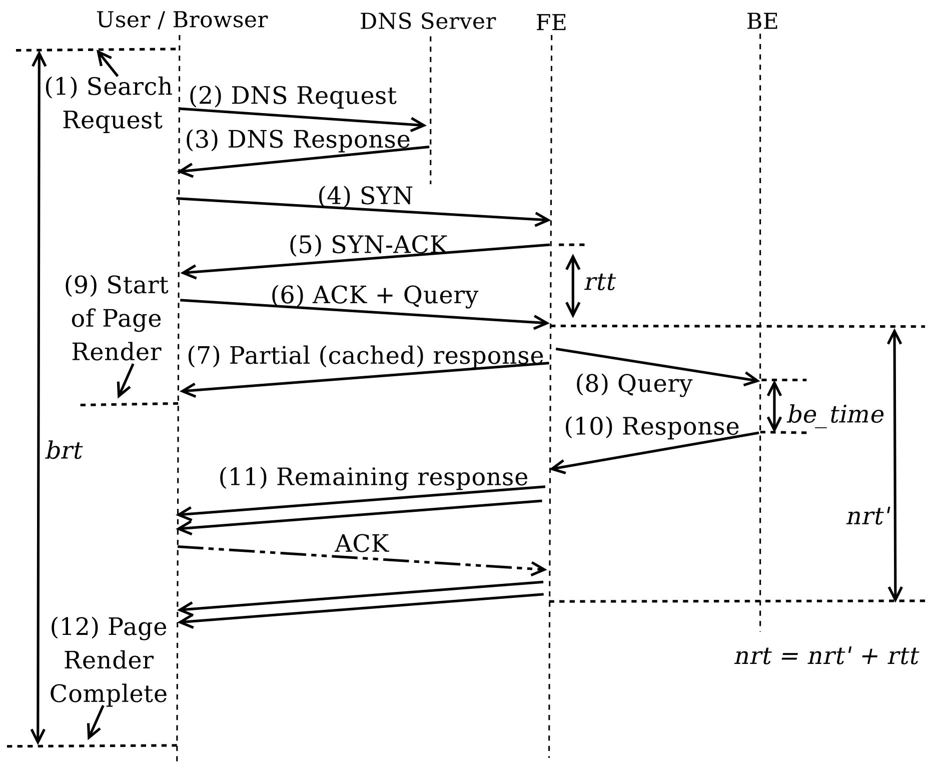


Figure 20: Messages and events during Web search transaction with Google. Latency of this transaction is measured as network-level response time (*nrt*) and browser-level response time (*brt*).

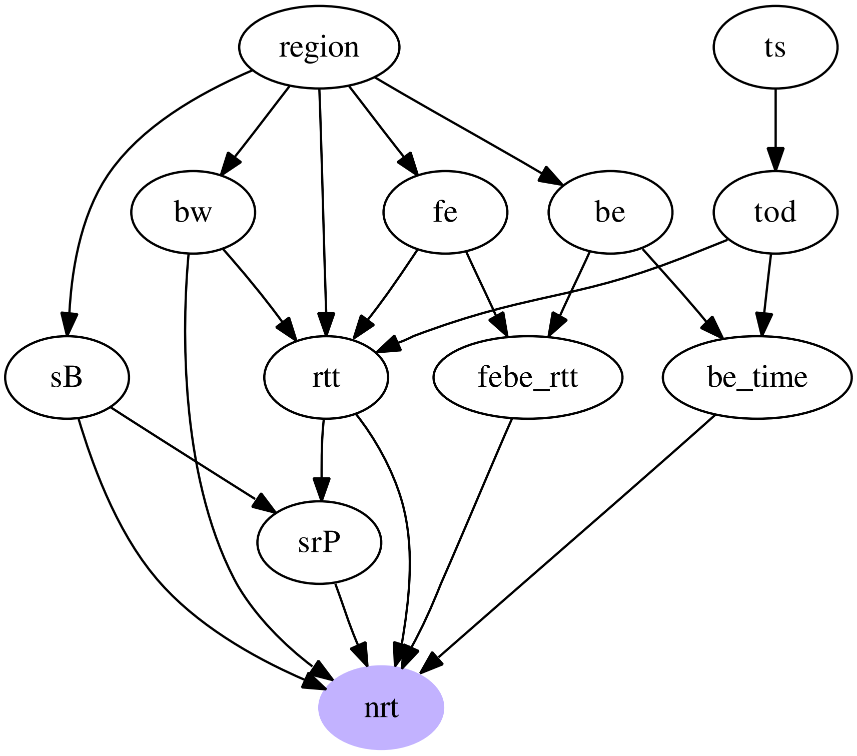


Figure 21: Inferred causal DAG for the dataset in Table 4. $A \rightarrow B$ means A *causes* B. *nrt* is target variable.

Table 7: Summary of causes to be addressed for achieving performance goals. For confidentiality, values of *be_time*, *rtt* and *febe_rtt* factors are presented relative to mean in the reference distribution.

Responsible variable		Percentage of high-latency transactions	Value of responsible variable in high-latency clusters		in reference distribution	
var. name	units	affected	mean	IQR	mean	IQR
(a) Causes to address to mitigate high-latency in a country.						
Transactions in the USA with latency over 80th percentile *nrt* in the USA.						
be_time	-	84	1.90	1.49–1.97	1.00	0.57–1.29
rtt	-	37	4.95	1.25–4.61	1.00	0.42–1.04
febe_rtt	-	19	2.22	1.19–2.38	1.00	0.24–1.19
sB	kB	13	28.6	22.2–30.4	9.0	6.8–9.0
Transactions in Australia with latency over 80th percentile *nrt* in Australia.						
be_time	-	70	1.91	1.42–2.18	1.00	0.49–1.39
rtt	-	47	3.86	2.03–3.74	1.00	0.21–1.53
srP	count	18	3.9	3.0–4.0	0.2	0.0–0.0
sB	kB	18	18.8	9.2–26.0	8.2	6.5–8.5
bw	kb/s	10	640	460–744	4277	978–6595
Transactions in India with latency over 80th percentile *nrt* in India.						
rtt	-	73	4.07	1.77–4.21	1.00	0.29–0.83
bw	kb/s	42	234	105–328	745	492–908
be_time	-	36	2.25	1.82–2.55	1.00	0.46–1.36
srP	count	17	4.0	2.0–4.0	0.4	0.0–0.0
sB	kB	13	23.9	8.7–27.0	8.6	6.9–8.4
(b) Causes inhibiting transaction latency comparable to USA.						
Transactions in Australia with latency over mean *nrt* in USA.						
febe_rtt	-	82	8.99	9.11–9.23	1.00	0.24–1.19
rtt	-	52	5.46	2.33–5.86	1.00	0.42–1.04
bw	kb/s	43	1131	730–1288	5938	2277–9069
Transactions in India with latency over mean *nrt* in USA.						
febe_rtt	-	96	6.30	6.79–6.97	1.00	0.24–1.19
rtt	-	91	5.59	1.48–6.23	1.00	0.42–1.04
bw	kb/s	87	662	447–864	5938	2277–9069
srP	count	14	3.5	2.0–4.0	0.1	0.0–0.0

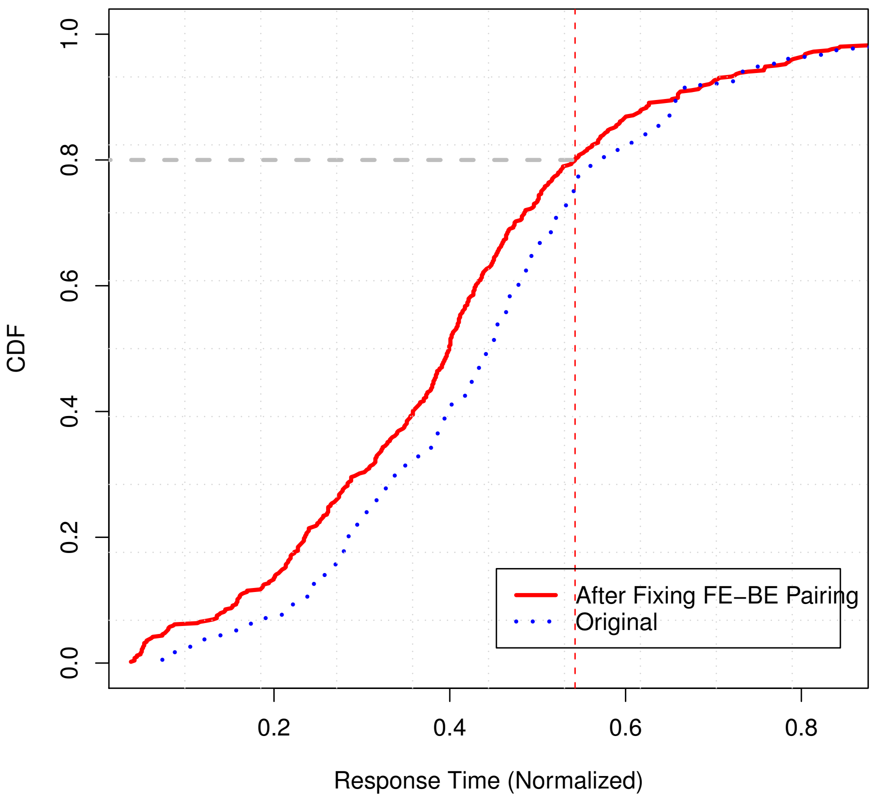


Figure 22: Predicted improvement in *nrt* for clients served from a FE that previously had bad FE-BE pairing.

CHAPTER IV

DETECTING NETWORK NEUTRALITY VIOLATIONS USING CAUSAL INFERENCE WITH NANO

This chapter presents *NANO*, a system that detects when ISPs apply policies that discriminate against specific classes of applications, users, or destinations. Existing systems for detecting discrimination are typically specific to an application or to a particular discrimination mechanism and rely on active measurement tests. Unfortunately, ISPs can change discrimination policies and mechanisms, and they can evade these tests by giving probe traffic higher priority. *NANO* detects ISP discrimination by passively collecting performance data from clients. To distinguish discrimination from other causes of degradation (*e.g.*, overload, misconfiguration, failure), *NANO* establishes a causal relationship between an ISP and observed performance by adjusting for confounding factors. *NANO* agents deployed at participating clients across the Internet collect performance data for selected services and report this information to centralized servers, which analyze the measurements to establish causal relationship between an ISP and performance degradations. We have implemented *NANO* and deployed clients in a controlled environment on Emulab. We run a combination of controlled experiments on Emulab and wide-area experiments on PlanetLab that show that *NANO* can determine the extent and criteria for discrimination for a variety of discrimination policies and applications.

4.1 Introduction

Network neutrality states that ISPs remain neutral to how they forward user traffic, irrespective of content, application, or sender [20]; ISPs may *discriminate* against certain subsets of users or services. Rather than taking a stance in this debate, we aim to make the policies of Internet service providers more *transparent* to end users, so that they can detect when ISPs degrade performance or connectivity for some subset of users or applications.

Because discrimination can take many forms, detecting it is difficult. ISPs have been interfering with TCP connections for BitTorrent and other peer-to-peer applications [18]; recently, British Telecom throttled video content [10] after previously demanding compensation from content providers such as BBC for increased traffic due to their content [9]; Cox Communications also said that it planned to begin throttling peer-to-peer traffic [3]. Other types of discrimination may include blocking specific ports, shaping traffic for specific services, or enforcing traffic quotas.

Existing mechanisms for detecting ISP discrimination actively probe ISPs to test for specific cases: Glasnost detects spurious TCP reset packets of BitTorrent connections [18], Beverly *et al.* study port-blocking [50], and NVLens detects prioritization by observing the type-of-service field in ICMP time-exceeded messages [57]. These tools detect specific classes of discrimination, but they have several drawbacks. First, they are *specific* to either the application (*e.g.*, BitTorrent) or the mechanism that the ISP is using to discriminate (*e.g.*, resetting TCP connections, or setting TOS bits). Second, they rely primarily on *active probes*, which are typically detectable, making it possible for an ISP to either block or prioritize them. Because discrimination may vary depending on the application or the mechanism, and ISPs can evade detection mechanisms that rely on active probes, users need detection tools that rely primarily on observations of *in situ* network traffic.

We present the design, implementation, and controlled evaluation of Network Access Neutrality Observatory (*NANO*), a system that infers the extent to which an ISP's policy causes performance degradations for a particular service. Instead of trying to determine whether an ISP is discriminating using a particular mechanism, *NANO* tries to infer whether there are differences in performance achieved through a particular ISP when compared to other ISP(s) for a given service. *NANO* tries to establish a causal relationship between an ISP's policy and the *observed* degradation of performance for a service using only passively collected data. Because *NANO* directly uses the observed performance of the service, it is difficult for ISPs to evade *NANO* inference, while at the same time discriminate against a service to degrade its performance. *NANO*'s techniques apply to general performance metrics and can thus apply to many services and applications. For example, throughput can be used to characterize the performance for both Web traffic (including pages, embedded content, video, etc.) and non-Web traffic (*e.g.*, FTP, BitTorrent). Similarly, jitter and loss rate can characterize the performance of many real-time services, such as interactive voice, video, or gaming traffic.

NANO's design draws inspiration from statistical epidemiology: Just as epidemiologists seek to determine whether a particular drug might be responsible for the improved health of a patient, we seek to determine whether a particular ISP affects performance degradation. The challenge in establishing causality is that many *confounding* factors may be the underlying cause for the observed outcome. Many factors other than the ISP may affect the performance of a particular service or application. For example, a service may be slow (*e.g.*, due to overload at a particular time of day). A service might be poorly located relative to the customers of the ISP. Similarly, a service may be fundamentally unsuitable for a particular network (*e.g.*, Internet connectivity is not suitable for VoIP applications in many parts of the world). Similarly, the performance might depend on software or hardware, or other network peculiarities.

NANO identifies when service performance differs across ISPs but confounding factors are equal. A big challenge in designing *NANO* is to identify the confounding factors and create an environment where all confounding factors are equal or

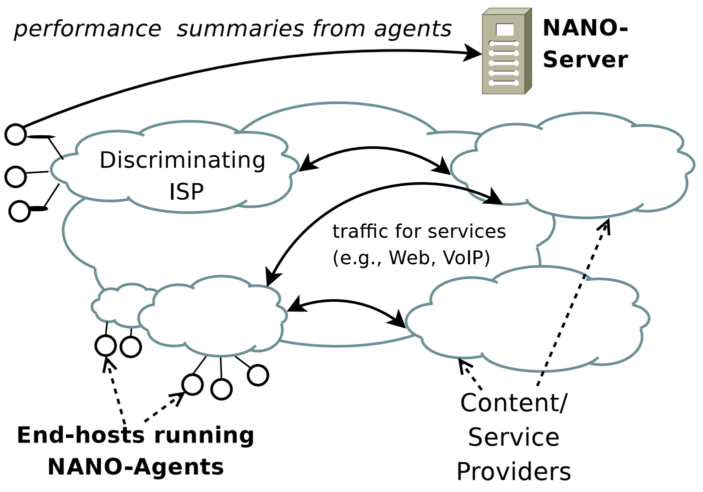

performance summaries from agents
NANO-Server
Discriminating ISP
traffic for services (e.g., Web, VoIP)
End-hosts running NANO-Agents
Content/ Service Providers

Figure 23: NANO architecture.

independent of the ISP or service performance. Although these goals are difficult to achieve, *NANO* can infer causal relationships by adjusting for confounding variables on passively observed data. Applying this approach has two main requirements: (1) enumerating the confounding factors and collecting data for the possible values of these variables, and (2) establishing a "baseline" level of service performance for a given set of values for the confounding variables that serves as a point of comparison. *NANO*'s client-side software, *NANO*-Agent, collects and reports performance data to *NANO*-Servers regarding their traffic, as well as various meta-data (*e.g.*, the CPU load on the machine at the time, the operating system, the type of connection, etc.) as shown in Figure 23. *NANO* then analyzes this performance data to quantify the causal relationship between an ISP's policy and the observed service degradation.

We have implemented the *NANO*-Agent and Server and made the *NANO*-Agent available for download [41]. We have evaluated *NANO* in a controlled environment; we emulate access network ISPs on Emulab, where clients perform HTTP and BitTorrent downloads from hundreds of PlanetLab nodes across the Internet; some ISPs in our setup discriminate while others remain neutral. We demonstrate that, even when the distribution of performance from the discriminating ISPs may look similar to the distribution of performance from the neutral ISPs, *NANO* detects discrimination, estimates the total causal effect on the performance of the services, and determines the discrimination criteria. Our goal in this chapter is to describe the *NANO* techniques and describe the implementation and controlled evaluation as a proof-of-concept. We do not yet have a sufficient deployment to infer ISP discrimination in real networks, but the NANO project Web site [41] provides the participating clients with other useful performance statistics. With a more extensive deployment, we hope to ultimately report on general discrimination practices across ISPs.

This chapter is organized as follows. Section 4.2 defines and motivates the problem, provides definitions, and articulates the challenges. Section 4.3 provides background on causal inference and formulates ISP discrimination detection as a statistical causal inference problem. Section 4.4 describes the design of *NANO* and Section 4.5 describes the implementation. Section 4.6 evaluates the accuracy, sensitivity, and scalability of *NANO*. Section 4.7 lists various open issues with *NANO*.

4.2 Problem and Motivation

Problem statement We aim to detect whether an ISP causes performance degradation for a service when compared to performance for the same service through other ISPs.

Definitions A *service* is an "atomic unit" of discrimination (*e.g.*, a group of users or a network based application). *Discrimination* is an ISP policy to treat traffic for some subset of services differently such that it causes degradation in performance for the service. Metrics for performance may be service-specific. We say that an ISP *causes* degradation in performance for some service (*i.e.*, that it discriminates against some service) if we can establish a causal relation between the ISP and the observed degradation. For example, an ISP may discriminate traffic for a particular application (*e.g.*, Web search), traffic for a particular domain, or traffic carrying particular type of media, such as video or audio, such that performance for these services degrades.

Challenges Detecting discrimination is challenging for the following reasons.

A. *The mechanism for discrimination may not be known.* Although existing tools for detecting network neutrality all assume that either the mechanism for discriminating against traffic or the application being discriminated against is known, this is generally not the case. Users often do not even know whether an ISP might be discriminating certain subsets of traffic. These users need methods for detecting discrimination that do not rely on testing for specific discrimination types.

B. *The baseline performance for a service in an ISP is not known.* Users do not know what the "baseline" performance is for a given service through their ISP, so detecting when the performance is degraded, potentially as a result of discrimination is difficult. We propose one approach to establish baseline performance in Section 4.4.2.

C. *Many factors can cause performance degradation.* Any tool that detects discrimination must identify the ISP—as opposed to any other possible factor—as the underlying cause of discrimination. An industry source recently expressed skepticism about the effectiveness of existing tools: "However, one ISP industry source, who asked not to be identified, questioned whether the tools would accurately point to the cause of broadband problems. 'Spyware or malware on computers can affect browser performance, and problems with the wider Internet can cause slowdowns, the source said."' [23]. It is precisely this problem—adjusting for such external causes—that we tackle.

We believe that *NANO* is the first technique that can isolate such discrimination from other confounding factors, without *a priori* knowledge of an ISP's discrimination policy. *NANO* relies on knowledge of confounding variables, but these are not difficult to enumerate using domain knowledge. *NANO* uses monitoring agents to collect values for the confounding variables.

4.3 Background

In this section, we give a brief overview of causal inference and how it can be used to quantify causal effect.

4.3.1 Causal Effect and Confounders

Statistical causal inference is applied in many observational and experimental studies [26, 45]. We review causal inference and describe how it relates to inferring ISP discrimination.

Causal effect *"X causes Y"* means that a change in the value of X (the "treatment variable") should cause a change in value of Y (the "outcome variable"). Accessing a particular service through an ISP is our treatment variable (X), and the observed performance of a service (Y) is our outcome variable: $X = 1$ when a user accesses a service through some ISP, and $X = 0$ when the user does not access the same service through that same ISP (*e.g.*, it accesses it through an alternate ISP). The value of the outcome variable Y depends on the service; it might be a direct measure of quality, such as Mean Opinion Score (MOS) for VoIP applications, or another variable that is highly indicative of the application performance, such as, throughput, loss-rate, or jitter.

Causal inference estimates the effect of the treatment variable (the ISP) on the outcome variable (the service performance). Let's define a *ground-truth* value for the outcome random variable as G_X, so that G_1 is the outcome value for a client when $X = 1$, and G_0 is the outcome value when $X = 0$. We refer to the outcome when not using the ISP ($X = 0$) as the *baseline*.

We can quantify the *average causal effect* of using an ISP as the expected difference of the ground truth of service performance between using the ISP and the baseline.

$$\theta = \mathbb{E}(G_1) - \mathbb{E}(G_0) \tag{3}$$

To compute the causal effect, θ, we must observe the outcome both under the treatment and without the treatment.

Association vs. causal effect In a typical *in situ* dataset, such as network traffic, each sample presents only the value of the outcome variable either under the treatment, lacking the treatment, but not both. Because the ground-truth values (G_0, G_1) are not simultaneously observable, we cannot estimate the true causal effect from an *in situ* dataset alone (Eq. 3). Instead, we can compute association. Let's define *association* as the difference of performance with or without the ISP:

$$\alpha = \mathbb{E}(Y|X = 1) - \mathbb{E}(Y|X = 0) \tag{4}$$

Of course, association is not a sufficient metric for causal effect, and in general $\alpha \neq \theta$, mainly due to the effects of confounding variables.

Confounding variables A *confounding variable* (or simply "confounder") is one that correlates *both* with the treatment variable in question (*i.e.*, the ISP) and the outcome variable (*i.e.*, the performance). Confounding variables make it difficult to assess the true extent of causal relationship between the treatment and outcome variable because if we observe a correlation between treatment and the outcome variables, we cannot be certain whether that correlation is because of a causal relationship between the treatment and outcome, or whether it is because of the confounding variable. For example, if the location of the client correlates with the ISP and the service, then we cannot blindly attribute any observed association between ISP and the service performance to the causal relationship between the two, because the difference in performance might be due to the differences in location of the ISP or services. The next section describes techniques for computing causal effect in the presence of confounding variables.

4.3.2 Dealing with Confounders

This section presents two techniques for estimating causal effect: random treatment and stratification. Stratification essentially emulates random treatment, albeit passively; we explain why stratification is more appropriate for network data.

Strawman: Random treatment If we assign clients to the treatment randomly, then under certain conditions, association is an unbiased estimator of causal effect.[1] All other variables that have an association with the outcome variable must remain fixed when the treatment changes. With *in situ* network traffic data, we must find a way to change the treatment variable—the user's ISP—while keeping other factors fixed. While random treatment is ideal for lab experiments, it is difficult to emulate on the Internet because it is difficult to make a user switch to an arbitrary ISP.

NANO approach: Stratification *NANO* uses a technique called stratification to adjust for confounding variables [26]; stratification places measurements into strata so that all samples in each stratum have "similar" values for the confounding variables, creating conditions that resemble random treatment: treatment and outcome variables become independent of confounding variables. Informally, one might think of this as placing all measurements where everything that could possibly be attributed to performance is equal, except for the ISP.

Stratification requires enumerating the confounding variables. Unfortunately, there is no automated way to enumerate all the variables that might affect an outcome variable; the problem is similar to any machine learning problem where accurate prediction depends on enumerating all of the features. Instead, we must rely on domain knowledge to enumerate the confounding variables and heuristics to identify

[1]This property holds because when X is independent of G_X, then $\mathbb{E}(G_X) = \mathbb{E}(G_X|X) = \mathbb{E}(Y|X)$; see [55, pp. 254–255] for a proof.

when some confounding variables may be missing. Although our evaluation (Section 4.6.2.3) shows that we have enumerated these confounders in our controlled environment, as clients become more diverse in a wide-area deployment, we may need to revisit and expand this list.

Formalization Let the *causal effect*, θ_{ij}, quantify how the performance of a service j, denoted by Y_j, changes when it is accessed through ISP i, versus when it is not accessed through ISP i. Let Z denote the set of confounding variables, and let s be a stratum. Then causal effect within a stratum, $\theta_{ij}(s)$ is:

$$\begin{aligned}\theta_{ij}(s;x) &= \mathbb{E}(Y_j|X_i = x, Z \in \mathbb{B}(s)) &(5)\\ \theta_{ij}(s) &= \theta_{ij}(s;1) - \theta_{ij}(s;0) &(6)\end{aligned}$$

where $\mathbb{B}(s)$ is the range of values of confounding variables in the stratum s. The variable $\theta_{ij}(s;0)$ in Equation 6 represents the *baseline* service performance. We can estimate the variance of $\theta_{ij}(s)$ (estimated as the variance of mean differences), as:

$$\sigma^2(\theta_{ij}(s)) = \frac{(n_{s1}-1)\sigma^2_{ijs1} + (n_{s0}-1)\sigma^2_{ijs0}}{n_{s0}+n_{s1}-2}\left(\frac{1}{n_{s0}} + \frac{1}{n_{s1}}\right) \tag{7}$$

where σ^2_{ijsx} is shorthand for $\sigma^2(\theta_{ij}(s;x))$ and n_{s1} and n_{s0} are the number of performance measurements that we observe with and without the ISP, respectively. Assuming that $\theta_{ij}(s)$ follows a Normal distribution, the (1-α) confidence interval is:

$$\hat{\theta}_{ij}(s) \pm z_\alpha \sigma(\theta_{ij}(s)) \tag{8}$$

Equations 6 and 8 estimate the causal effect for *each* stratum s. We would like to summarize the overall causal relationship between ISP i and a service j across *all* the strata. Unfortunately, all strata may not be equally likely and causal effect might vary across strata. The techniques used in epidemiology for this purpose (*e.g.*, the Mantel-Haenszel statistic [26]) only work for binary outcome variables and also require estimates of the prevalence of each outcome, so they are not appropriate in this context. Instead, we simply average across the strata as:

$$\hat{\theta}_{ij} = |s|^{-1} \sum_{s} \theta_{ij}(s) \tag{9}$$

If we assume that causal effect is independent across strata, we can estimate the variance and the corresponding (1-α) confidence intervals as:

$$\begin{aligned} \sigma^2(\theta_{ij}) = & \ |s|^{-2} \textstyle\sum_s \sigma^2(\theta_{ij}(s)) & (10) \\ (1-\alpha) \text{ CI for } \theta_{ij} & = \hat{\theta}_{ij} \pm z_\alpha \sigma(\theta_{ij}) & (11) \end{aligned}$$

The units for causal effect are same as for service performance, so the values are straightforward to interpret: essentially, for metrics such as, throughput, a negative value indicates performance degradation, and for metrics such as loss or jitter, a positive value indicates performance degradation.

4.4 NANO Approach

This section enumerates the confounding variables required for this inference and explains how NANO performs causal inference.

4.4.1 Confounders for Network Performance

In this section, we enumerate three categories of confounding variables and explain why they might correlate with both the treatment variable (the ISP) *and* the outcome (the performance). NANO-Agents collect statistics for variables that help determine the level of various confounding factors. Table 8 shows these variables.

Client-based confounders The application that a client uses for a service can affect service performance. Certain Web sites may be optimized for a particular Web browser, or differences in Web browsers may affect performance; for example, Opera, Firefox, and Internet Explorer use a different number of simultaneous TCP connections, and only Opera uses HTTP pipelining by default. Other features that may affect performance include the operating system and the configuration of the

Table 8: Data collected by NANO.

Client-Based Features
IP address
Process for each flow
CPU usage
Memory usage
Operating System
Uptime
Network devices and counters
Network-Based Features
From network traffic
IP address, port number, and protocol
Timestamps for first, last packet for each flow
Cumulative, periodic bytes, packets per flow
SYN/SYN-ACK SYN-ACK/ACK RTTs
TCP state for active connections
TCP retransmits
TCP duplicate ACKs
Provided by user
ISP contract level/SLA
Geographic Location
User Location (home/work/etc.)
Type of link (wireless/Ethernet)
Whether using a home router
Type of router

client's computer and local network, as well as a client's service contract. These variables clearly affect performance, but they may also correlate with the ISP. For example, we expect that Microsoft Windows may be more popular among home users that other operating systems, while Unix variants may be more common in academic environments. Similarly, certain browsers may be more popular among certain demographics and localities. If the ISPs cater different demographic groups, then these variables may correlate with the ISP brand.

Network-based confounders Various properties of the Internet path, such as the location of the client or ISP relative to the location of the servers, can degrade service. A path segment to a particular service provider might not be sufficiently provisioned. If we wish to disregard these effects, we must adjust for the path properties. We do not treat congestion as a confounder because we believe that there is value in determining whether the performance degradation related to congestion are specific to an ISP. NANO cannot differentiate between network-wide congestion that affects all the services and a congestion that is targeted at a particular service because NANO does not compare performance across services.

Time-based confounders Service performance varies widely with time of day, due to changes in utilization, and ISPs may also experience different performance at different times of day.

4.4.2 Establishing Causal Effect

NANO uses the five steps in Figure 24 to estimate causal effect of an ISP for service degradation. First, NANO stratifies the service performance data reported from the NANO-Agents. Next, NANO estimates the extent of possible causal effect of ISP on performance within each stratum by comparing the performance within the stratum with the baseline performance from other ISPs. Then NANO summarizes the causal effect by aggregating on all the strata and finally tests whether the aggregate causal effect is statistically significant. Optionally, NANO can also infer the criteria that the ISP is using for discrimination.

Step 1: Stratifying the data To stratify the data, NANO creates "bins" (*i.e.*, ranges of values) for each of the confounding variables. The main criteria for stratification is to create bins on the value of the confounding variable such that the value of

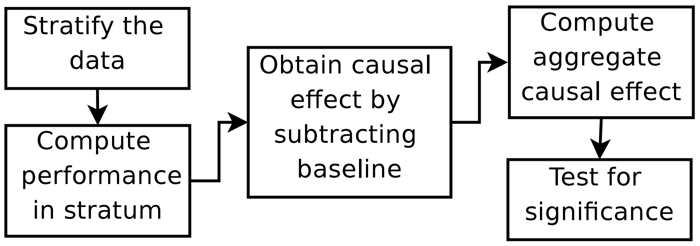

Stratify the data
Compute performance in stratum
Obtain causal effect by subtracting baseline
Compute aggregate causal effect
Test for significance

Figure 24: Steps for computing causal effect.

the treatment and outcome variables can be considered *independent* of the confounding variable for the span of the bin. As a result, the bin size depends on the nature of the confounding variable.

For discrete variables, creating bins is simple: we create strata such that there is a bin for every unique value of the variable. For example, all the clients using a particular version of the browser are in one stratum. For continuous variables, the bins must be sufficiently small such that the variable has essentially a constant value within the stratum. For example, many network performance metrics are correlated with the time-of-day variable due to diurnal cycles, but these metrics are typically independent of the time-of-day across days for a given hour [54], or time-of-day and day-of-the-week combined [33]. This characteristic makes one-hour bin a reasonable choice for time-of-day variable. If the performance cycles repeat only on weekly basis, then we may additionally stratify on a variable representing the day-of-the-week.

We apply standard correlation tests to determine whether the treatment variable and the outcome variable are independent of the confounding variable within a stratum. To reduce the number of strata and the overall number of samples needed to establish confidence intervals, we combine adjacent strata if the distribution of the outcome variable conditioned on the treatment variable is identical in each stratum.

Step 2: Computing causal effect We compute causal effect by plugging in the performance estimates for each stratum in Eq. 6. One challenge with using Eq. 6 is establishing the baseline performance (term $\theta_{i,j}(s;0)$ in Eq. 6). Intuitively, this value reflects the performance a user would see *without* treatment (*i.e.*, not using an ISP i for some service). Simply using a different ISP is insufficient if that ISP is *also* discriminating against service j.

To address this problem, NANO computes the baseline, $\theta_{i,j}(s;0)$, as the average service performance when not using ISP i; *i.e.*, the average over all other ISPs that have users in that stratum:

$$\theta_{i,j}(s;0) = \sum_{k \neq i}^{n_p} \theta_{k,j}(s;1)/(n_p - 1)$$

where $n_p > 2$ is the number of ISPs for which we have clients in stratum s. One limitation of this definition is that if many ISPs are discriminating against a service, the baseline performance will reflect discrimination (*i.e.*, discrimination becomes the norm). In such cases, we could derive the baseline in other ways, such as by comparing against the best performance instead of the average, or by using a performance model of the service from laboratory experiments or mathematical analysis.

Step 3: Inferring the discrimination criteria Although NANO's causal inference makes no assumptions about discrimination criteria used by the ISP, NANO can determine discrimination criteria as a side effect of stratification. NANO infers the discrimination criteria that an ISP uses by using simple decision-tree based classification methods. For each stratum and service where NANO detects discrimination, NANO assigns a negative label, and for each stratum and service where it does not detect discrimination, it assigns a positive label. NANO then uses the values of the confounding variables and the service identifier as the feature set and the discrimination label as the target variable, and a decision-tree algorithm to train the classifier. The rules that the decision tree generates indicate the discrimination criteria that the ISP uses because the rules indicate the boundaries of maximum information distance between discrimination and normal behavior.

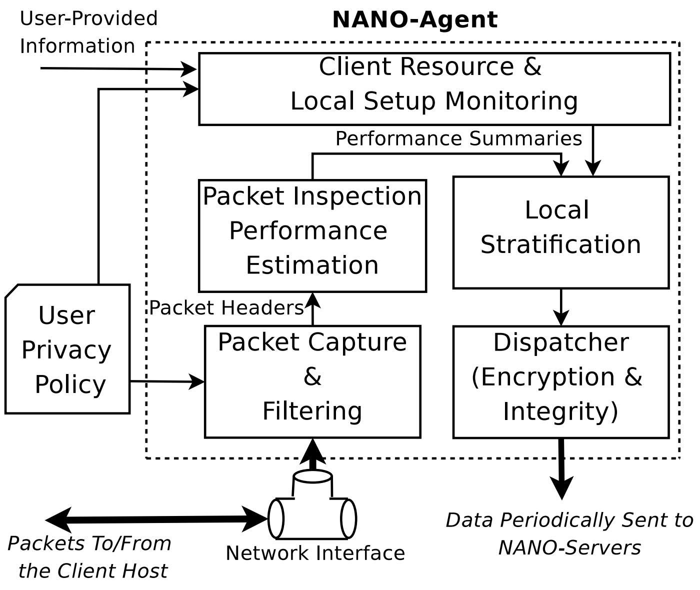

User-Provided Information
NANO-Agent
Client Resource & Local Setup Monitoring
Performance Summaries
Packet Inspection Performance Estimation
Local Stratification
Packet Headers
User Privacy Policy
Packet Capture & Filtering
Dispatcher (Encryption & Integrity)
Packets To/From the Client Host
Network Interface
Data Periodically Sent to NANO-Servers

Figure 25: Design of client-side agent for NANO.

4.5 Design and Implementation

This section describes the implementation of NANO, which has two parts: *NANO-Agents* and a *NANO-Server*. NANO-Agents reside on participating clients and continuously monitor and collect data for traffic from that client host to various destinations, and send aggregate traffic statistics to the centralized NANO-Server. The NANO-Server collects these statistics and performs the inference described in Section 4.3 to quantify the ISP's effect on performance. The primary source of data for NANO are client-side agents installed on computers of voluntarily participating clients (NANO-Agents). We describe these components in detail below.

4.5.1 Agents

We have developed NANO-Agent as a packet-level sniffer, running at clients, that can access fine-grained information from the client machines including the various system resource utilization and client machine setup information.

Figure 25 shows the architecture for the NANO-Agent. After a packet is captured from the network interface (via pcap), the modules shown in Figure 25 strip the protocol headers and compute performance summaries for the traffic. The agent only computes performance summaries for traffic that is allowed by a user-specified privacy policy. The agent periodically dispatches the summaries to NANO-Server. The NANO-Agent collects three types of features for the confounding factors, corresponding to the three classes of confounding variables described in Section 4.4.1 and Table 8.

Data collection The NANO-Agent analyzes the network and transport protocol headers to identify the service and assess performance. For the experiments that we describe in Section 4.6, we focus on features from the TCP/IP headers of the packets and associated timing information. We try to estimate the throughput and latency that the packets experience for a TCP flow. To estimate the throughput, the agent continuously measures the bytes uploaded and downloaded in a specified (configurable) interval. To estimate the latency on a TCP flow, the agent measures the latency between the SYN and SYN-ACK packets for the flows that originate at the client, and the latency between the SYN-ACK and the subsequent ACK, for the incoming connections that the client receives. The agent keeps track of connection duration and events such as losses, timeouts (by tracking the TCP duplicate acknowledgments) or unexpected connection terminations, such as, with a TCP reset flag. Our implementation also infers the application associated with the active flows from the `proc` file system. In addition, the agent also continuously monitors the average CPU and memory utilization on the client host.

In addition to runtime statistics, the agent also collects information about the client setup which includes the client host specification, including the platform, CPU and memory specification, and the active operating system on the host. NANO relies on the user to provide the agent with information that it cannot infer. This information includes the type of network interface (wired or wireless), the type of contract the client has with the ISP, and the user's location (city and country). In the future, we plan to use an IP-to-geographic location database in lieu of client-provided information. Table 8 describes the variables that the NANO-Agent collects.

Protecting user privacy Information that NANO-Agents collect may contain sensitive information (*e.g.*, destinations a client has visited or the amount of content it has downloaded). Protecting user privacy is paramount. Unfortunately, standard anonymization techniques, such as anonymizing IP addresses and various other features (*e.g.*, browser type, operating system) obfuscate the very features used to stratify the data, thus preventing causal inference. We must apply techniques that mask client identities but preserve the features that NANO uses to stratify the data. NANO employs three measures that protect user's private data from eavesdropping, allows user's some control over the type of traffic that NANO-Agents monitor, and reduce the granularity of information that is sent to the NANO-Server. These measures mitigate privacy concerns that arise due to passive monitoring, although these concerns are not completely alleviated.

1. Local stratification. The NANO-Agent performs local stratification to reduce the granularity of information that the client sends to the server. The NANO-Agent can optionally report only the /24 prefix of the source and destination IP addresses, allowing some obfuscation of client identity, and similarly round off the round-trip time measurements to (configurable) nearby values.

2. User-specified filters. The NANO-Agent allows the users to specify a set of fully qualified or wild-card domain names, IP addresses, or port numbers (or ranges) that they wish to have excluded from the monitoring. NANO-Agent does not collect information about the flows matching the filters. We are working on allowing users to specify filters based on application names as well, so that the user can, for example, specify whether to prevent BitTorrent traffic from being monitored. Users can also temporarily disable NANO-Agent; during this period, NANO-Agent does not monitor any traffic but continues to send beacon packets to the server indicating that it is alive but not monitoring.

3. Secure communication. NANO-Agents transmit data to the servers over SSL. The SSL certificate is included with the NANO-Agent distribution.

Implementation We have implemented the NANO-Agent using C++ for Linux-based systems. The agent promiscuously captures the packets that flow from the client host's network interface using the `pcap` library.

The NANO-Agent implements user privacy filters as follows. If the user specifies a source or destination IP address or port ranges, the filtering is straightforward: NANO ignores packets for corresponding flows. For the filters using domain names, the NANO-Agent inspects the DNS traffic at the client and learns the IP addresses for the domain names that match the filters. The NANO-Agent then ignores the traffic with the matching IP addresses for the duration of TTL specified in the DNS. The NANO-Agent keeps track of CNAME records and ignores traffic for IP addresses mapped to canonical names of the domain names in the filters. For example, if the user specifies `mail.google.com` as one of the filters, then the NANO-Agent would also ignore packets with IP addresses for `googlemail.l.google.com`, as it is (sometimes) returned as canonical name for `mail.google.com`.

The NANO-Agent uses protocol buffers [22] to maintain the information that it collects. Protocol buffers offer high speed and compact serialization that allows us to minimize the computational and communication overhead for running the agent at the client. The agent periodically serializes the data that it has collected and sends it to a NANO data collection server. The NANO-Agent implementation is open-source and freely available [41].

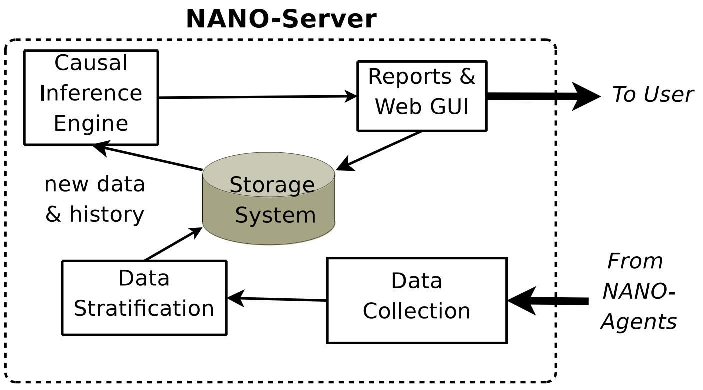


Figure 26: NANO-Server design.

4.5.2 Server

The NANO-Server periodically receives information from NANO-Agents running on the clients and performs the causal analysis. Although some of the causal inference logic is currently offline, the NANO-Server provides a Web-based interface [41] where participating users can observe live statistics only about the traffic summaries that their NANO-Agents send to the NANO-Server. We are working on making these statistics more useful, including providing real-time diagnostics information, and network "health" monitoring that can be inferred from the data that the NANO-Agent collects. Additionally, we allow users to selectively delete their data from NANO-Servers.

Implementation We have implemented the server using a combination of C++ to implement the data collection and demarshalling and using a Python and MySQL back-end for analysis and causal inference. Our implementation can compute causal effect over 20,000 strata in about one minute using two threads on a dual 3.2 GHz processor Intel Pentium 4 machine with 4 GB of memory. In a real-world deployment, the number of strata can easily approach in millions; for this, we plan to port the NANO-Server to a Map-Reduce-based implementation [16].

4.6 Evaluation

This section presents the results of our evaluation. The experimental setup is as shown in Figure 27 and a summary of experiments that we conduct is presented in Table 9. We study the following four questions in our evaluation:

- *Can NANO detect different types of discrimination?* (Section 4.6.2.1) To answer this question, we tested NANO's detection algorithms with three different types of discrimination and in the presence of various network and client-side confounding variables.
- *Can NANO determine discrimination criteria?* (Section 4.6.2.2) Our evaluation shows that NANO can determine the discrimination criteria used by the ISP using a decision-tree based classifier.
- *Can NANO determine when confounders are missing?* (Section 4.6.2.3) Our experiments show that NANO's heuristic for testing sufficiency of confounders is a reasonable one, although this problem in general is an open question.
- *How does NANO scale with the size of the input data?* (Section 4.6.2.4) We study how NANO's accuracy is affected by the amount of input data, and how memory and CPU requirements scale with the size of the input.

4.6.1 Experiment Setup

Tested: PlanetLab and Emulab We use PlanetLab [8] nodes as servers. Specifically, we configured a set of geographically distributed PlanetLab nodes with two types of services. First, we configure two Web servers on each of these PlanetLab nodes to represent two different Web services. Second, we have configured the PlanetLab nodes to act as BitTorrent clients.

We use Emulab to create a set of ISPs, each with its set of clients that connect to the ISP using links of configurable characteristics. The ISP provides connectivity to the Internet to its clients. Figure 27 shows this arrangement. The clients in the Emulab environment access these services through emulated ISPs where we can introduce discrimination using Click routers. The clients can be configured with different physical configurations and run different operating systems on them. This setup allows us to control some of the confounding variables on the client side and use various discrimination criteria that might be implemented by an ISP. Each client also runs an instance of NANO-Agent which periodically reports the performance data to a NANO-Server running at Georgia Tech. We did not gather samples from all ISPs for all strata, so we only consider the strata where at least three of the five ISPs in our experiment had 20 samples or more. As a result, the baseline performance for some of the strata might comprise fewer than four ISPs. For the experiment involving discrimination against long flows, 96% of strata met the other criteria; this figure was even higher for the other experiments.

A potential problem with this setup is that if an ISP in the wide-area (outside our Emulab environment) is discriminating against traffic between the Emulab clients and the PlanetLab nodes, then NANO would not be able to detect that, since NANO only has data from the NANO-Agents which all use the ISP outside of Emulab. Indeed, we encountered paths to PlanetLab nodes that were very lossy to begin with. However, because we have no reason to believe that the losses on these paths correlate with the ISPs that we emulate on Emulab, these paths are not a confounder for our experiments, and we can afford to ignore these even if the ISPs on those paths are potentially discriminating.

Emulating discrimination We emulated ISP discrimination by running Click on the Emulab node that acts as the ISP router to connect the ISP clients to the Internet (see Figure 27). We pass the client traffic through the Click router running as a kernel module on the router node. We used a combination of Click elements to perform various forms of discrimination including probabilistically dropping packets on all flows,

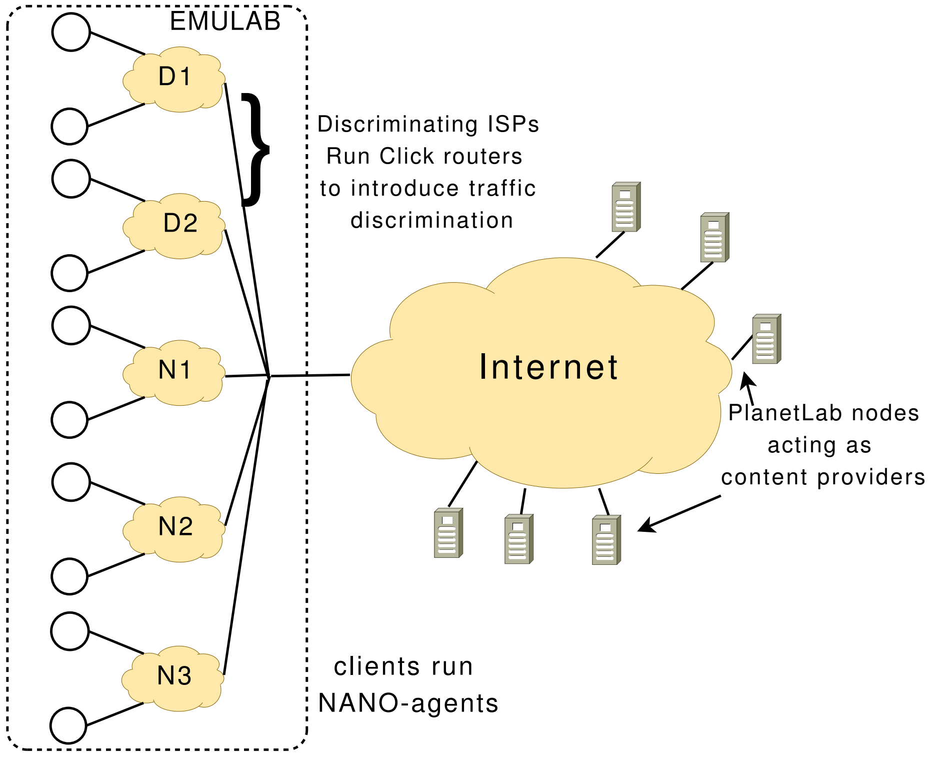


Figure 27: Experimental setup on Emulab. Each ISP is an Emulab node with a Click router that performs different types of discrimination, depending on the experiment. Clients run NANO-Agents and report information to a NANO-Server.

or flows which exceed a certain length, dropping of TCP acknowledgments, dropping packets for a particular service or destination, and sending TCP RST packets back to the client (similar to the practice by Comcast).

We used available Click router elements such as `IPClassifier` to classify packets based on the various IP and TCP fields, `RandomSample` for dropping packets and, `AggregateIPFlows` and a modified version of `AveragePktCounter` to classify flows that exceed a certain length. Running Click router as a kernel module was sufficient to ensure that the Emulab ISP node could sustain a reasonable amount of traffic (maximum of 20 Mbps).

Figure 28 shows an example of discrimination against long flows that we implemented using a Click router. In this case, we configured the router to probabilistically

Table 9: Summary of NANO experiments.

Internet Service Providers (ISPs): ISPs N_1, N_2, and N_3 are neutral. ISPs D_1 and D_2 discriminate.

Experiment 1. Simple Discrimination. ISPs D_1 and D_2 discriminate against HTTP traffic for all clients. ISPs D_1 and D_2 drop 0.1% and 0.3% of the packets respectively. Location of the HTTP server is a confounding variable. ISPs N_1, N_2, N_3, D_1, and D_2 access the content from the *near* PlanetLab servers with probabilities, 0.4, 0.1, 0.7, 0.6, and 0.9, respectively, and access the *far* PlanetLab servers with the remaining probability.

Experiment 2. Long Flow Discrimination. ISPs D_1 discriminates against S_1, and D_2 discriminates the HTTP traffic for S_2 for all their clients *if* the flows from S_1 or S_2 exceeds certain limits. ISPs D_1 and D_2 drop 0.1% and 0.3% of the packets for flows exceeding 10,000 and 13,000 packets respectively.
Server location is a confounder, as in Experiment 1, with same probabilities for the *near* HTTP servers. All HTTP servers provide both S_1 and S_2, albeit on different ports.

Experiment 3. BitTorrent Discrimination. ISP D_2 discriminates the BitTorrent traffic for all its clients if the BitTorrent peer is not in certain subset of PlanetLab nodes. dropping 0.3% of the packets of the flows that are established with the non-preferred peers.

drop TCP packets of flows that exceeded 13,000 packets. For the flow shown in Figure 28, this event occurs at around 10 seconds after the start of the flow, at which point a drop in both throughput (calculated as bytes transferred per ten seconds) and the rate of cumulative packets for the flow occurs.

4.6.2 Results

This section presents our experimental results; we answer the questions that we posed at the beginning of Section 4.6.

4.6.2.1 Can NANO detect discrimination?

We performed three experiments to evaluate whether NANO could detect discrimination. We create five ISPs: two of these discriminate, and we call them discriminating

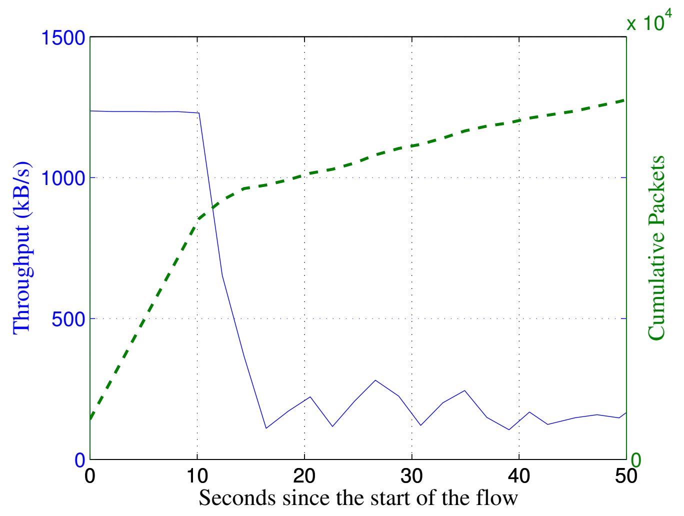
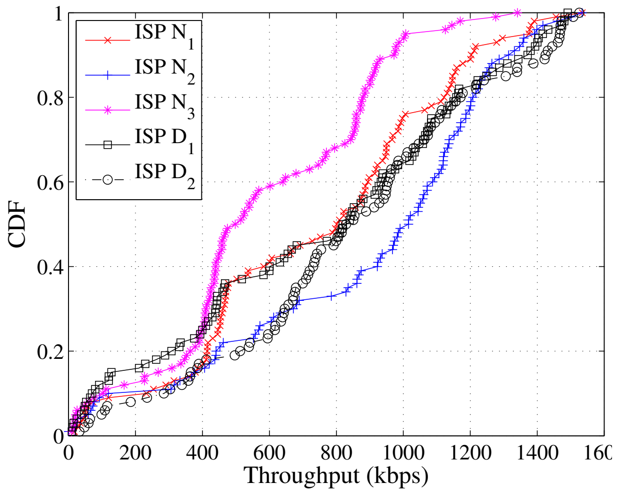


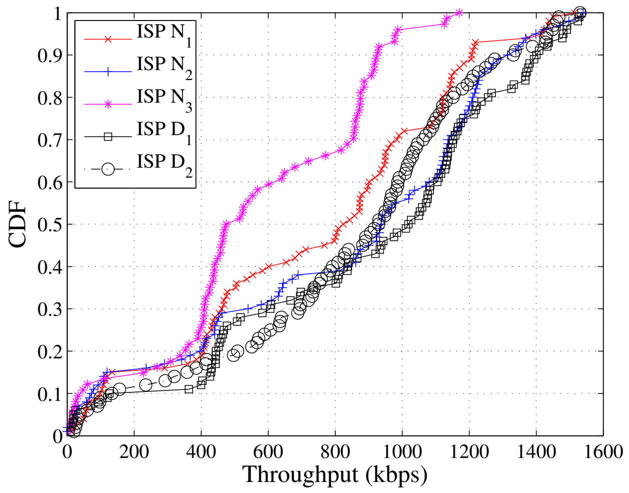
Figure 28: Throttling throughput for long TCP flows using Click modular router.

ISPs, ISP D_1 and ISP D_2. The remaining three ISPs use best-effort service for all the packets on their routers; we refer to these as the neutral ISPs and ISP N_1, ISP N_2, and ISP N_3. Table 9 summarizes these experiments. The first is a simple discrimination against HTTP traffic; the second is a discrimination against long-running flows (in practice, this might be bulk transfers like movies); the third involves discrimination against BitTorrent traffic.

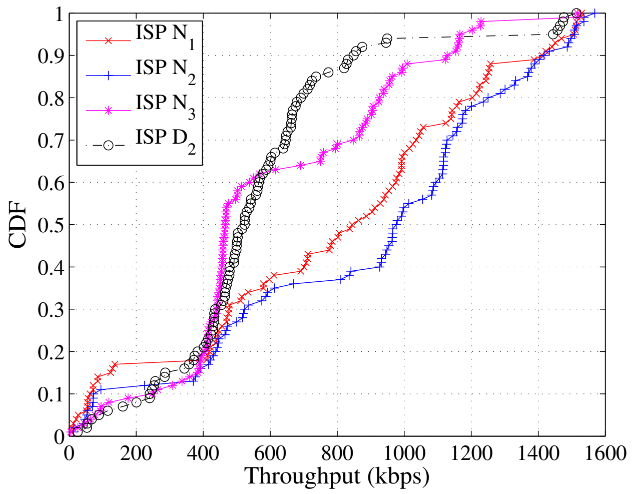
It is not possible to detect discrimination from distributions alone Figure 29 shows the distribution of performance for the clients of each ISP in all three experiments. For the first two experiments, we compute the average throughput over the life of the flow and use that as a metric. Note that it is difficult to detect which ISP is discriminating solely based on the performance distribution. For example, in the first two experiments (Figures 29 (a) and (b)), the overall performance distribution for the discriminating ISPs is similar or better than the distribution of performance in the neutral ISPs because the clients in the discriminating ISPs access the *near* servers with higher probability. Because geographically closer servers are more likely to provide higher throughput, the discriminating ISPs still have a larger fraction of sessions with higher throughput. Similarly, the throughput for most BitTorrent clients in D_2



(a) Simple Discrimination.



(b) Long Flow Discrimination.



(c) BitTorrent Discrimination.

Figure 29: Performance distribution for the clients in all the ISPs. Although ISP D_1 and ISP D_2 are discriminating in each of the three scenarios, it is not possible to correctly identify the ISPs that are discriminating or the extent of discrimination by observing the over the performance distributions.

is similar to the throughput in the other ISPs when the ISP is discriminating against a subset of destinations.

NANO detects discriminating ISPs and quantifies the extent of discrimination We present the average causal effect on throughput in kbps with a 90% confidence interval (cf. Equation 11) for each ISP, service, and experiment in Table 30. (Due to the number of strata, we do not have enoug space space to list the causal effect of each ISP for each strata.)

For the neutral ISPs, N_1, N_2, and N_3, we find that the adjusted difference in performance compared to average is slightly positive in all experiments; *i.e.*, the performance of the service is somewhat better than average for these ISPs. If we had a large number of ISPs, we would have expected the average causal effect to be exactly zero (*i.e.*, the performance using these neutral ISPs should be equal to the baseline). Because the experiment has only five ISPs (three neutral and two discriminating), the performance from the discriminatory ISPs decreases the baseline performance, which in turn causes the performance of the neutral ISPs to appear slightly above baseline. The 90% confidence interval for each of the neutral ISPs spans both sides of zero (neutral).

For the discriminating ISPs, D_1 and D_2, NANO finds negative causal effect on throughput in each of the experimental scenarios where these ISPs discriminated. For example, the confounding adjusted causal effect for ISP D_1 is -108 kbps for the Simple Discrimination experiment, indicating that the throughput is 108 kbps less than the baseline throughput; the entire confidence interval is in the negative range.

Table 30 also shows that NANO also avoids mischaracterizing an ISP when it is not discriminating. In the Long Flow Discrimination experiment, ISP D_1 does not discriminate against service S_2; similarly, ISP D_2 does not discriminate the flows for service S_1: NANO correctly estimates close to neutral performance for these ISPs in these cases. On the other hand, in the cases where these ISPs discriminate, NANO correctly detects a negative effect on throughput. In the third experiment involving BitTorrent, NANO computes a negative causal effect for ISP D_2, which degrades performance for BitTorrent flows that are not in a preferred set of subnets. The causal effect on the throughput is about 300 kbps below the baseline throughput performance.

The confidence intervals for discriminating ISPs are wider than the confidence intervals for the neutral ISPs because when the discriminating ISPs drop packets for TCP flows, the throughput for these flows becomes more variable. The confidence intervals for the experiment involving BitTorrent are also larger than other experiments involving discrimination, which probably results from three causes. First, BitTorrent downloads smaller chunks at a time, and such transfers can be bursty and have high variance in throughput. Second, BitTorrent uses more simultaneous connections, which interfere with each other and increase variance. Finally, BitTorrent's peer selection criteria means that there may not be a steady transfer between a pair of peers. We also observe that in general the causal effect of ISP D_2 is more negative than ISP D_1, which occurs because ISP D_2 uses a higher packet drop rate (0.3%) than ISP D_1 (0.1%). As a result, more degradation occurs in ISP D_2 than in D_1.

4.6.2.2 Can NANO determine discrimination criteria?

We evaluate the extent to which NANO can determine the discrimination criteria for each of the three experiments.

To infer the criteria that ISP D_1 is using to discriminate against long flows, we labeled the stratum for service S_1 where we detected more than 100 kbps of causal effect as the discriminated strata and the remaining strata as undiscriminated. We then ran the J.48 decision tree on this labeled dataset, where the data columns included the /24 subnet (`location`) of servers and average number of cumulative packets (`cum_pkts`) for a session. The decision tree produces the following rules:

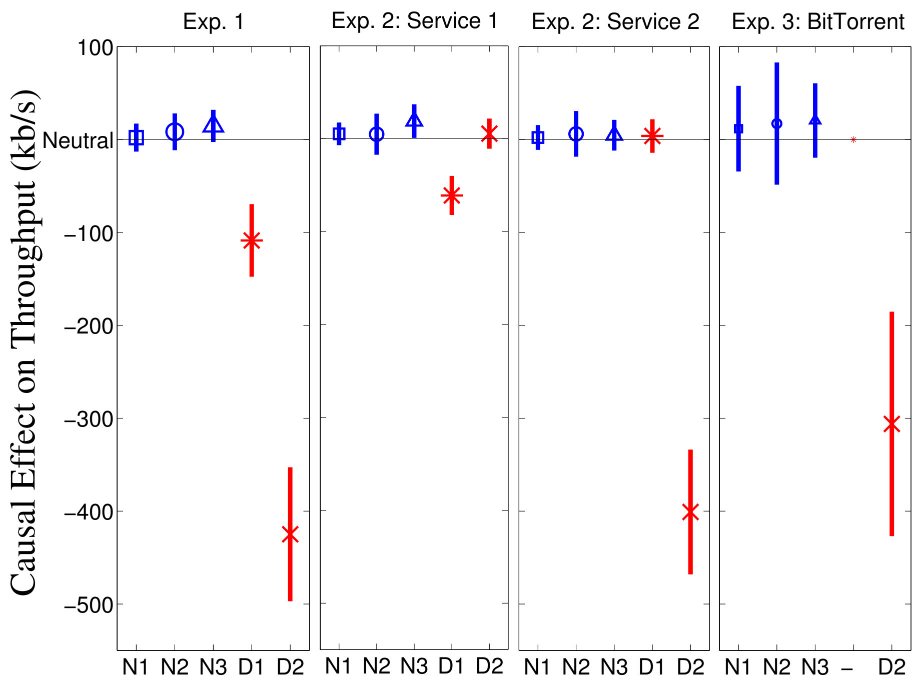


Figure 30: Causal effect (in kbps) for each ISP using Eq. 11, with 90% confidence intervals. Each point indicates the confounding-adjusted average difference in throughput for each service and ISP in various experiments; the lines extend to 90% confidence intervals. Near-zero values imply that the performance is close to *baseline* and there is no observed causal relationship between the ISP and the service. Large negative values indicate a significant causal relationship between the ISP and performance degradation. Table 10 presents numeric values for the causal effect and confidence intervals.

```
cum_pkts <= 10103 --> not_discriminated

cum_pkts > 10103 --> discriminated
```

and yields 89% accuracy with a 7% false positive rate. The decision tree ignored the `location` variable, correctly inferring that ISP D_1 is not discriminating based on destination but rather when the flow's duration exceeds 10,103 packets. Recall from Table 9 that D_1 drops packets for flows exceeding 10,000 packets.

For the BitTorrent experiment, we labeled the strata similarly, and used the J.48 algorithm. The resulting decision tree correctly identifies most of the identifiers for

Table 10: Causal effect (in kbps) for each ISP using Eq. 11, with 90% confidence intervals. Figure 30 presents values for the causal effect and confidence intervals graphically.

ISP	Causal Effect on Service Performance (kbps)	
Experiment 1. Simple Discrimination		
	HTTP Service	
N_1	2.10±15.1	
N_2	8.39±19.8	
N_3	14.65±17.3	
D_1	**-108.6±39.1**	
D_2	**-424.91±72.1**	
Experiment 2. Long Flow Discrimination		
	HTTP S_1	HTTP S_2
N_1	5.17±12.2	2.20±13.4
N_2	4.80±17.3	6.1±24.6
N_3	18.9±18.1	4.65±16.5
D_1	**-61.20±21.0**	3.82±18.1
D_2	5.36±16.2	**-400.91±67.2**
Experiment 3. BitTorrent Discrimination		
	BitTorrent	
N_1	11.71±46.1	
N_2	17.2±65.8	
N_3	20.56±40.1	
D_2	**-306.13±120.8**	

the /24 networks that the ISP D_2 discriminates. The accuracy and false positive rates are 76% and 14% respectively.

We used the same technique for ISP D_1 in the Simple Discrimination experiment, but the decision tree fails to produce a conclusive tree, which is expected because D_1 discriminates against all sessions without any criteria.

4.6.2.3 Can NANO identify sufficiency of confounders?

There is no automated way to enumerate all the confounding variables or determine that a passive dataset has all the confounding variables. We apply a heuristic that helps determine whether we are missing any major confounding variables.

If we are capturing all the confounding factors, then a regression function $f(X; Z)$, trained on the ISP, X, and the confounding factors Z, should accurately predict the response time y, and the predicted value $\hat{y}$, should be unbiased. We can test for bias by verifying that the distribution of error, $(y - \hat{y})/y$, is centered at zero, with high confidence, and that there is no correlation between the error and the outcome variable. Additionally, if the confounders are proximate or direct (other) causes for service performance, $f()$, should be able to predict the outcome variable. There may be other causal variables besides the ISP and the confounders that influence performance and are needed for high-accuracy prediction. However, as long as these variables are not correlated with the ISP, we do not need to account for them. As a result, the predictor $f()$ does not have to be high accuracy to rule out missing confounders, it only needs to be unbiased.

We analyze this heuristic for determining the performance for service S_1. We used the ISP with the network round-trip time, the relative location of clients and servers, and the cumulative bytes as the input variables for the predictor, $f()$. We use one random 2/3 of the data for this experiment as training and other 1/3 as test, and predict the performance for the test data. We found that the error was centered at zero and showed small correlation with the outcome value. The correlation values between the error and outcome variable are 0.01, 0.08, 0.05, 0.01, and 0.05 for ISPs N_1, N_2, N_3, D_1 and D_2, respectively, thus indicating that it is unlikely that there is a major confounding variable missing.

Figure 31 shows the modulo prediction error, $|(y - \hat{y})/y|$. The median error is less than 30% for all the ISPs. Between 10% and 30% of the measurements have 60% or more error for various ISPs, indicating that there may be other causes for performance degradation besides what we capture. Because the error does not seem to correlate with either the ISP or the performance, we believe that those causes do not correlate with ISP and performance at the same time, and are thus not confounding.

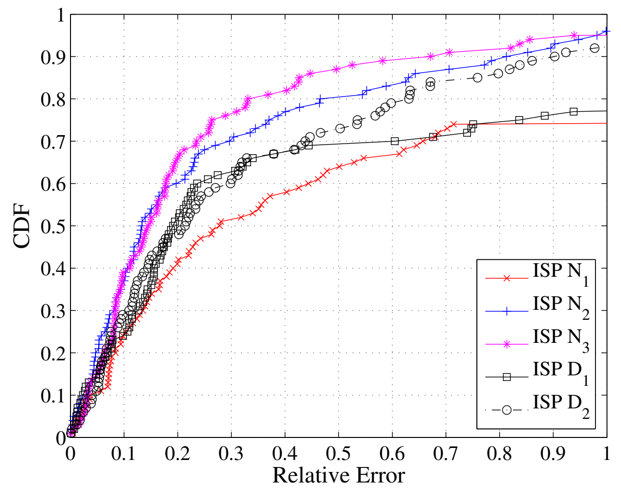


Figure 31: Distribution of relative error in predicting the throughput for the Long Flow Discrimination experiment with the variables that NANO collects.

4.6.2.4 Does NANO scale with the size of the input data?

We discuss the effect of data volumes on accuracy and the effect of number of clients on NANO server load.

Data vs. accuracy As shown in Equation 6 and Equation 11, the confidence level on the number of session measurements from the discriminating and the neutral ISPs, as well as the number of strata in which we observe data.

The results in Table 30 for Experiment 2 are obtained using about 1,900 strata (121 for location, and 15 for cumulative packets in a flow—other confounding factors did not have variability for this experiment) and about 100,000 periodic measurements obtained from the NANO-Agents. To assess the effect of fewer measurements, we performed two random sub-samples of 50,000 and 20,000 measurements each. We re-calculated the causal effect, and found that the mean causal effect did not change appreciably for any of the ISPs, but the confidence interval widened by 1.2–1.9 times for various ISPs using 50,000 measurements, and by 1.8–2.7 times for various ISPs using only 20,000 measurements; this conforms to our expectations of having confidence intervals expand by a factor of $\sqrt{5}$ and $\sqrt{2}$, respectively. After a participating client installs the NANO-Agent, NANO servers receive data continually, which makes it easy to collect several hundred thousand measurements for every client in a matter of few days (We collected the measurements for the experiments in roughly 2 hours). Thus, in a real deployment, NANO should be able to gather enough measurements to be highly accurate.

The other aspect of accuracy is *coverage*; the more strata that NANO-Agents cover, the better the information it can provide information about discrimination based on certain features. Certain confounders, such as time-of-day variation, are easy to cover because they simply require gathering estimates over a long time interval without requiring additional clients to participate. Increasing coverage across other confounders, such as location or application type, the type of network interface, or operating system, require participation from additional clients. The required number of clients grows roughly as the size of the cross-product of the range of the confounding variables that require separate clients for measurements. We note that NANO-Agents running on laptops may be quite valuable for helping to increase coverage, because they can cover multiple locations, ISPs, and network-interface types as the user roams. NANO tracks the changes. Unfortunately, as the user roams, some of the information that the user provides to NANO-Agent at installation time becomes invalid. We are improving NANO so that it can automatically infer these variables instead of relying on users to provide the correct values.

CPU and storage overhead We present some back-of-the-envelope numbers on the scalability of NANO. Based on the data that NANO-Server is receiving from the early adopters, uncompressed reports require about 3.2 kB every ten seconds on average per user. Using an estimate of 2.5 kbps per user, to support 10,000 users, the NANO-Servers need roughly 25 Mbps bandwidth and about 12 GB of storage per day to archive the user reports. On a server with 3.2GHz CPU and 4 GB of RAM, NANO takes about 22 ms on average to demarshall a client report, stratify the data, and insert it into a database. At this rate, NANO can support about 5,500 clients in real-time. Our current implementation spawns a new process for batch processing of all the reports that the server receives every minute. Optimizing this process can improve scalability further. Since NANO-Agents perform a DNS lookup to establish a connection to the NANO-Server, we could ultimately use DNS load balancing to distribute load and storage across multiple servers.

4.7 Discussion

In this section, we address limitations with *NANO* and our ongoing work to address these limitations.

Sufficiency of confounding variables for real world application If *NANO* fails to adjust for certain confounding variables, it may miscalculate the causal effect: both over and underestimation are possible. Unfortunately, there is no automated way to enumerate all confounding variables for a problem, or to conclusively test that a given set of confounding variables is sufficient. As in biostatistics and epidemiology where passive datasets are used for causal inference, enumerating confounding variables relies on domain knowledge. Fortunately, network performance is well understood among researchers, and it is relatively easy to enumerate the variables that correlate with ISPs and can also significantly affect service performance. We believe that the variables that we listed is comprehensive, and any remaining confounding variables will have only minimal effect on accuracy of causal inference. Still, if we find that there are additional confounding variables, we can collect data for them in updated releases for *NANO*-agents and correct the inference at the server end.

Better privacy for users In addition to the techniques already implemented in *NANO*-Agents, *NANO*-Agents could further protect user privacy in three ways. First, *NANO*-Agents could collect data from only the top ‘k’ Web sites (*e.g.*, according to Alexa) and strip personally-identifiable information from the payloads of this traffic. The data collected from clients would only reveal whether they had visited popular sites (not overly sensitive, since many users visit these sites) and the performance they were experiencing to those sites. Second, *NANO* could use a combination of passive and active measurements to produce the corpus of data used for inference; in such cases, the *NANO*-Server would receive all measurements, but would not be able to distinguish which traffic was generated solely to probe the network and which traffic was actually initiated by the client. Finally, clients using *NANO* might send their reports through an anonymizing network (*e.g.*, Freenet [14]) that obfuscates the source of the original report. Of course, the IP addresses of the clients would still be contained in the traffic traces, but in the process of mixing, IP addresses on various traces could possibly be swapped without affecting the stratification.

Integrity of reports from agents *NANO*-Agents could lie about the data they collected, by producing false traces or modifying the statistics about the traffic at the client. In these cases, it may be difficult to detect when a client reports false statistics about its observed network performance. We suggest two possible techniques that could help mitigate this possibility. First, *NANO* could collect data from *NANO*-Agents that have similar values for various confounding factors (*e.g.*, same upstream ISP, same portion of the network topology). In these cases, reported performance measurements yield continuous discrepancies, *NANO* could determine that a *NANO*-Agent was reporting inaccurate results.

Defense against evasion *NANO* establishes causality by measuring the difference in expected values for response times given the use of a specific ISP and its deviation from baseline measurements. An ISP might try to conceal discrimination by treating traffic such that *mean* value of performance remains unaffected. For example, ISP may give exceptionally good performance to some clients and degrade performance for others. To defend against this type of attack, we imagine that *NANO* might be extended in two ways. First, we could modify *NANO*'s causal inference algorithms to operate on *multiple points in the response-time distribution*, as opposed to simply inferring causality based on mean values. Second, presuming that an ISP's attempt to game the detection may vary over a range of time, we could run *NANO*'s inference algorithm over different time granularity to attempt to catch more fine-grained variations in an ISP's policies across users, services, or applications.

Augmenting with active measurements *NANO*'s reliance on passive measurements is limiting because *NANO* can obtain measurements for specific services only when the user of the client accesses those services. On demand or active measurements from the client can be beneficial in a number of ways; (a) measurements to known good services can help mitigate chances of false inference; (b) if there are not enough samples for comparison between ISPs for certain stratum or services, these samples can be obtained actively. *NANO*-Agent has an active probing client that periodically contacts *NANO*-Server to obtain information about servers for which active measurements are desired. Currently, *NANO*-Agent can perform HTTP GET and POST to servers that the *NANO*-Server directs the client to using HTTP redirects.

Motivating users to install *NANO*-Agents *NANO* users can only draw meaningful conclusions if they compare their measurements to those of other users. Thus, encouraging users to deploy *NANO* is critical to its success. To provide users an immediate incentive to install *NANO*-Agents, independently of whether other users have deployed agents, we have developed a Web interface where users can view their own performance statistics in isolation, as well as compare their performance statistics to other users. We hope that allowing users to analyze their own data through an interactive interface will give users the incentive to deploy the tool and help build a critical mass of users running *NANO*-agents.

CHAPTER V

RELATED WORK

In this section we provide an overview of prior work related to this dissertation.

A key component in response time for Web requests is TCP transfer latency. There has been significant work on TCP throughput and latency prediction using TCP modeling [4, 44, 11]. Due to inherent complexity of TCP these models make simplifying assumptions to keep the analysis tractable; these assumptions may produce inaccurate results. Recently, there has been effort to embrace the complexity and using past behavior to predict TCP throughput. He et al. [25] evaluate predictability using short-term history, and Mirza et al. [39] use machine-learning techniques to estimate TCP throughput. We also use machine-learning and statistical inference in our work, but techniques of [39] are not directly applicable because they rely on estimating path properties immediately before making a prediction. Further, they do not provide a framework for evaluating *what-if* scenarios. The parametric techniques, as we show in Section 2.8.5, unfortunately are not very accurate for predicting network-level response time. Further, TCP transfer is not sufficient for predicting browser-level response time.

Like WISE, WebProphet [34] also uses a dependency structure to estimate impact of changes on page-load time. Unlike WISE, which uses network measurements as nodes in the causal dependency graph, WebProphet uses Web-page objects as nodes. Further, WebProphet only models simple, linear dependencies in latency, whereas WISE uses a full-blown non-parametric model that can estimate arbitrary functions. We believe this technique is promising, but requires collecting client-side measurements, which may not always be available to a CDN operator. As we show in Section 2.8.5 WISE can accurately predict browser-level response time using only server-side logs.

Recent work has used Bayesian inference for fault and root-cause diagnosis. SCORE [31] uses spatial correlation and shared risk group techniques to find the best explanation for observed faults in the network. Shrink [27] extends this model to a probabilistic setting, because the dependencies among the nodes may not be deterministic due to incomplete information or noisy measurements. Sherlock [5] additionally finds causes for poor performance and also models fail-over and load-balancing dependencies. Rish et al. [49] combine dependency graphs with active probing for fault diagnosis. These projects focus on diagnosis but do not evaluate *what-if* scenarios.

Perhaps the most closely related work to HIP is WhyHigh [32], which focuses on helping network operators identify groups of prefixes that are experiencing high latency for Google's Web service. WhyHigh seeks to explain the underlying causes of high round-trip time for groups of clients. This information is useful, but operators also need to know the causes for high latencies in service transaction times. As Figure 21 shows, high round-trip time is only one possible cause for high latencies in service response times. Our results in Table 7 shows that other factors, such as backend processing time, client access network bandwidth, loss and retransmits cause a significant number of high-latency transactions.

HIP relies on causal discovery to identify potential causes of high latency transactions; there is a large body of related work on finding root causes of performance and reachability problems in networks. Several recently developed tools use techniques based on Bayesian belief propagation or minimum set to find root causes for network-related problems. SCORE [31] uses spatial correlation and shared risk group techniques to find the best explanation for observed faults in the network. Shrink [27] extends this model to a probabilistic setting, because the dependencies among the nodes may not be deterministic due to incomplete information or noisy measurements. Sherlock [5] finds causes for poor performance and also models fail-over and load-balancing dependencies. NetDiagnoser [17] and EtherTrace [37] use minimum set cover based approaches to isolate root causes for reachability problems in ISP networks; these tools focus on locating failures within a network, not on isolating groups of users with performance problems. Rish et al. [49] combine dependency graphs with active probing to improve fault diagnosis.

The challenges in HIP are more difficult that those that any of these systems address. With the exception of Sherlock, existing techniques diagnose problems for binary failures. Sherlock considers considers a third "troubled" state for causes, but does not analyze performance for clients in terms of a continuous response-time distribution like HIP. On the other hand, HIP analyzes causes of high latency; thus, both the contributing factors for Web transaction latency and the output variable itself are continuous; both of these characteristics make the problem more challenging than existing approaches.

WebProphet [34] uses dependencies between Web-page objects to predict a client's browser-level response time. WebProphet requires collecting client-side measurements, which may not always be available to a CDN operator. Still, if this data is available, we believe that HIP's techniques might also apply to WebProphet to help operators identify how to improve client download times for specific Web pages.

Glasnost [18] detects TCP connection resets of peer-to-peer applications. It simulates the BitTorrent protocol and detects spurious TCP RST packets which might be generated by the ISP to throttle BitTorrent. Glasnost is effective at detecting current discrimination technique against BitTorrent traffic, but if ISPs change the discrimination mechanism, then Glasnost will need to incorporate new tests. NVLens [57] focuses on detecting performance degradation among backbone ISPs via setting of TOS bits in the IP headers of the ICMP packets, so its analysis is specific to this mechanism.

Both Network Diagnostic Tool (NDT) [12] and Network Path and Application Diagnostics (NPAD) [38] rely on active client probing to detect network performance issues. Netalyzr [42] performs a series of tests using the client's browser to check the status of commonly used protocols, such as, POP, Bittorrent, and SSH. Diffprobe [28] performs active measurements from client machines to M-Lab [40] nodes to detect any ISP traffic discrimination based on traffic shaping mechanisms. These tools perform active measurements, which are detailed, but also evadable.

Tripwire uses a fingerprint-based technique to detect modification of in-flight packets [48]. We focus on violations that result in performance degradation, rather than modification of content. These techniques rely on active measurements and focus on specific discrimination mechanisms as opposed to *in situ* measurements.

NetDiff [36] detects performance differences between backbone ISPs. NetDiff uses the geographic location as a normalizing factor for fair comparison between ISPs, and in a sense adjusts for a confounding factor in the assertion that one ISP is better than another. NetDiff uses ICMP packets to probe the paths, but an ISP can evade detection by detecting such probe packets. *NANO* overcomes these difficulties by passively monitoring the performance of the various services. *NANO* builds on previous work on characterizing ISP networks [35] and monitoring ISP SLAs [51] to adjust for ISP topology differences. Keynote [30] compares performance across backbone ISPs; in addition to the above drawbacks, it also requires ISP cooperation for placement of measurement nodes, which may not be possible.

Another interesting point in the design space is comparison performance of similar services within an ISP and then determining whether there is a difference in performance among these services; NVLens [57], for instance, compares the latency for BitTorrent packets with the performance for HTTP packets. We believe that while interesting, this choice of comparison presents additional challenges that can be difficult to overcome. The services may differ in ways that naturally affect their performance: for example, a service that sends packets at a higher burst-rate may experience higher loss and latency for similar average transfer rate. Even if two services have similar traffic patterns, (*e.g.*, due to both using TCP), the completion time for a request may depend on additional server side variables, such as the back-end delays, caching rate, or rate-limiting at the server end. A fair direct comparison between performance would require comparing the performance of services that are *similar* in all aspects that can affect a service's performance; this can be difficult to achieve in general.

CHAPTER VI

CONCLUSION

This dissertation presented tools and techniques for modeling performance of Internet-based services using probabilistic causal modeling. We presented three tools, WISE, HIP and NANO that use causal models to facilitate performance related network management tasks using causal reasoning on passively collected data. While these models are not a replacement a domain expert, they are extremely useful for automating complex tasks that arise routinely for network management. In particular, we presented tools for three important network management problems.

Answering What-if Questions. Network designers must routinely answer questions about how changes to configuration and deployment will affect service response times. Without a rigorous method for evaluating these scenarios, the network designers must rely on ad hoc methods or resort to costly field deployments. Chapter 2 presented WISE, a tool for evaluating *what-if* deployment scenarios for content distribution networks. To our knowledge, WISE is the first tool to automatically derive causal relationships from Web traces and apply statistical intervention to predict networked service performance. Our evaluation demonstrates that WISE is both fast and accurate: it can predict response-time distributions in "what if" scenarios with high accuracy. WISE is easy to use; its scenario specification language makes it easy to specify complex configurations in just a few lines of code. WISE is also deployed: Googlehas used WISE to evaluate what-if scenarios in practice.

Answering How-to Questions. One of the most vexing problems that content distribution network operators face is how to improve performance for their clients. Much recent work has focused on helping operators answer hypothetical "what if" configurations; these tools help operators determine how a particular configuration would affect client response time or performance, but they presume that an operator knows an appropriate hypothetical configuration to test in the first place. Thus, figuring out how to help operators answer these complementary “how to” questions has remained an important practical problem.

Chapter 3 presented HIP, the first tool to help CDN operators improve the performance of CDNs for clients. HIP performs clustering to help operators focus on groups of clients that are experiencing poor performance due to the same underlying case. It also performs temporal analysis to distinguish different types of causes that can help narrow the search for the appropriate corrective actions. Finally, it performs statistical intervention using an existing “what if” scenario evaluator, WISE, to evaluate the effectiveness of potential solutions.

Many of the techniques from HIP may apply to other networking problems where operators commonly encounter “how to” problems, such as traffic engineering and network troubleshooting. They may also ultimately help users of other existing “what if” tools improve client performance in those respective settings.

Quantifying ISP Discrimination. Network neutrality is an important problem for Internet-based services. The service providers and users want to know whether their ISP is degrading the performance in some way. Chapter 4 presented *NANO*, a tool that applies causal inference to passively collected data from end-hosts to detect service degradation caused by ISP discrimination. In contrast to existing approaches, which apply rules to detect specific types of discrimination and actively probe ISPs to detect discrimination, *NANO* observes *in situ* traffic and performs causal inference to determine whether the characteristics of the actual application traffic itself shows any variations that can be attributed to the ISP. *NANO*’s approach enables it to detect more types of discrimination than existing approaches and makes it more difficult for ISPs to evade by treating test probes differently than data traffic.

Our evaluation showed that *NANO* can detect discrimination for different policies and application types, thus demonstrating that *NANO*'s detection is general and can detect discrimination even when the ISP's discrimination policies are not known *a priori*, as long as the variables that may significantly confound the relationship between ISP policy and service performance are known. *NANO* can also quantify the extent of discrimination and identify the discrimination criteria. Our experiments also showed that *NANO* scales well with the number of clients. We have released *NANO* and intend to collect data from a wide range of heterogeneous *NANO*-Agents across the Internet. We are working with a large content provider to deploy *NANO* on a large, geographically distributed measurement platform.

REFERENCES

[1] "Hadoop." `http://hadoop.apache.org/core/`, 2008. (Date accessed: Jan. 2008).

[2] Akamai, "http://www.akamai.com," 1999. (Date accessed: Jan. 2008).

[3] Andersen, N., "Cox ready to throttle P2P, non "time sensitive" traffic." `http://tinyurl.com/bcexla`, Jan. 2009. (Date accessed: Jan. 2009).

[4] Arlitt, M., Krishnamurthy, B., and Mogul, J., "Predicting Short-transfer Latency from TCP arcana: a Trace-based Validation.," in *Proc. ACM SIGCOMM Internet Measurement Conference*, (New Orleans, LA), Oct. 2005.

[5] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M., "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Proc. ACM SIGCOMM*, (Kyoto, Japan), Aug. 2007.

[6] Barbir, A., Cain, B., Nair, R., and Spatscheck, O., *Known Content Network (CN) Request-Routing Mechanisms.* Internet Engineering Task Force, July 2003. RFC 3568.

[7] Barroso, L. A., Dean, J., and Holzle, U., "Web Search for a Planet: The Google Cluster Architecture," No. 2, pp. 22–28, 2003.

[8] Bavier, A., Bowman, M., Culler, D., Chun, B., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M., "Operating System Support for Planetary-Scale Network Services," in *Proc. First Symposium*

on Networked Systems Design and Implementation (NSDI), (San Francisco, CA), Mar. 2004.

[9] "Its back to 'Pipes' and 'Free rides': Internet neutrality under attack (again)." `http://tinyurl.com/lyjo98`, June 2009. (Date accessed: June 2009).

[10] "BT Heavily Throttling BBC, All Video." `http://tinyurl.com/m2v7f5`, May 2009. (Date accessed: May 2009).

[11] Cardwell, N., Savage, S., and Anderson, T., "Modeling tcp latency," in *Proc. IEEE INFOCOM*, (Tel-Aviv, Israel), Mar. 2000.

[12] Carlson, R., "Network Diagnostic Tool." `http://e2epi.internet2.edu/ndt/`. (Date accessed: July 2008).

[13] Cheng, D., Kannan, R., Vempala, S., and Wang, G., "A Divide-and-Merge Methodology for Clustering," *ACM Transactions on Database Systems*, vol. 31, no. 4, pp. 1499–1525, 2006.

[14] Clarke, I., "A distributed decentralised information storage and retrieval system," Master's thesis, University of Edinburgh, 1999.

[15] Collins, J., Hall, N., and Paul, L., "Do all and only causes raise the probabilities of effects?," 2002.

[16] Dean, J. and Ghemawat, S., "MapReduce: Simplified data processing on large clusters," in *Proc. 6th USENIX OSDI*, (San Francisco, CA), Dec. 2004.

[17] Dhamdhere, A., Teixeira, R., Dovrolis, C., and Diot, C., "Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in *Proc. CoNEXT*, Dec. 2007.

[18] Dischinger, M., Mislove, A., Haeberlen, A., and Gummadi, K. P., "Detecting bittorrent blocking," in *Proc. Internet Measurement Conference*, (Vouliagmeni, Greece), Oct. 2008.

[19] "Emulab." `http://www.emulab.net/`. (Date accessed: Jan. 2008).

[20] Felten, E., "Three Flavors of Net Neutrality." `http://www.freedom-to-tinker.com/blog/felten/three-flavors-net-neutrality`, Dec. 2008. Date accessed (Dec. 2008).

[21] Freedman, M. J., Freudenthal, E., and Mazières, D., "Democratizing content publication with Coral," in *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, (San Francisco, CA), Mar. 2004.

[22] "Protocol Buffers." `http://code.google.com/apis/protocolbuffers`. (Date accessed: Dec. 2008).

[23] Gross, G., "Google, partners release net neutrality tools." `http://www.thestandard.com/news/2009/01/28/google-partners-release-net-neutrality-tools`, Jan. 2009. (Date accessed: Jan. 2009).

[24] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Ed.* Springer, 2009.

[25] He, Q., Dovrolis, C., and Ammar, M., "On the predictability of large transfer tcp throughput," in *Proc. ACM SIGCOMM*, (Philadelphia, PA), Aug. 2005.

[26] Jewell, N., *Statistics for Epidemiology.* Chapman & Hall/CRC, 2004.

[27] Kandula, S., Katabi, D., and Vasseur, J.-P., "Shrink: a tool for failure diagnosis in ip networks," in *ACM SIGCOMM workshop on Mining network data (MineNet'05)*, (New York, NY, USA), pp. 173–178, ACM, 2005.

[28] Kanuparthy, P., "Diffprobe: Detecting ISP Traffic Discrimination." `http://www.cc.gatech.edu/~partha/diffprobe/`. (Date accessed: Nov. 2009).

[29] Kaufman, L. and Rousseeuw, P. J., *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.

[30] "Keynote Home Page." `http://www.keynote.com/`, 1999. (Date accessed: Jan. 2009).

[31] Kompella, R. R., Yates, J., Greenberg, A., and Snoeren, A. C., "Ip fault localization via risk modeling," in *Proc. 2nd USENIX NSDI*, (Boston, MA), May 2005.

[32] Krishnan, R., Madhyastha, H. V., Jain, S., Srinivasan, S., Krishnamurthy, A., Anderson, T., and Gao, J., "Moving beyond end-to-end path information to optimize CDN performance," in *Proc. Internet Measurement Conference*, 2009.

[33] Lambert, D. and Liu, C., "Adaptive thresholds: Monitoring streams of network counts," in *Journal of the American Statistical Association*, vol. 101, No. 473. Applications and Case Studies, Mar. 2006.

[34] Li, Z., Zhang, M., Zhu, Z., Chen, Y., Greenberg, A., and Wang, Y.-M., "Webprophet: Automating performance prediction for web services," in *Proc. 7th USENIX NSDI*, (San Jose, CA), Apr. 2010.

[35] Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T. E., Krishnamurthy, A., and Venkataramani, A., "iPlane: An information plane for distributed services," in *Proc. 7th USENIX OSDI*, (Seattle, WA), Nov. 2006.

[36] Mahajan, R., Zhang, M., Poole, L., and Pai, V., "Uncovering Performance Differences among Backbone ISPs with Netdiff," in *Proc. 5th USENIX NSDI*, (San Francisco, CA), Apr. 2008.

[37] Mansy, A., bin Tariq, M., Feamster, N., and Ammar, M., "Characterizing VLAN-Induced Sharing in a Campus Network," in *Proc. Internet Measurement Conference*, (Chicago, Illinois), Oct. 2009.

[38] Mathis, M., Heffner, J., and Reddy, R., "Network Path and Application Diagnosis." `http://www.psc.edu/networking/projects/pathdiag/`. (Date accessed: Jan. 2009).

[39] Mirza, M., Sommers, J., Barford, P., and Zhu, X., "A machine learning approach to tcp throughput prediction," in *Proc. ACM SIGMETRICS*, (San Diego, CA), June 2007.

[40] "Measurement Lab." `http://measurementlab.net`, Jan. 2009. (Date accessed: Jan. 2009).

[41] "NANO Website.." `http://www.gtnoise.net/nano`. (Date accessed: Dec. 2008)

[42] "Netalyzr." `http://netalyzr.icsi.berkeley.edu/`. (Date accessed: Jan. 2009).

[43] Netezza, "Business intelligence data warehouse appliance." `http://www.netezza.com/`, 2006. (Date accessed: Jan. 2008).

[44] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proc. ACM SIGCOMM*, (Vancouver, British Columbia, Canada), pp. 303–323, Sept. 1998.

[45] Pearl, J., *Causality: Models, Reasoning, and Inference.* Cambridge University Press, 2000.

[46] Pike, R., Dorward, S., Griesemer, R., and Quinlan, S., "Interpreting the data: Parallel analysis with sawzall," *Scientific Programming Journal: Special Issue on Worldwide Computig Programming Models and Infrastructure*, vol. 13, pp. 227–298, Dec. 2005.

[47] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[48] Reis, C., Gribble, S. D., Kohno, T., and Weaver, N. C., "Detecting in-flight page changes with web tripwires," in *Proc. 5th USENIX NSDI*, (San Francisco, CA), Apr. 2008.

[49] Rish, I., Brodie, M., and Ma, S., "Efficient fault diagnosis using probing," in *AAAI Spring Symposium on Information Refinement and Revision for Decision Making: Modeling for Diagnostics, Prognostics, and Prediction*, 2002.

[50] Robert Beverly, S. B. and Berger, A., "The internet's not a big truck: Toward quantifying network neutrality," in *Passive & Active Measurement (PAM)*, (Louvain-la-neuve, Belgium), Apr. 2007.

[51] Sommers, J., Barford, P., Duffield, N., and Ron, A., "Efficient Network-wide SLA Compliance Monitoring," in *Proc. ACM SIGCOMM*, (Kyoto, Japan), Aug. 2007.

[52] Sprites, P. and Glymour, C., "An algorithm for fast recovery of sparse causal graphs," in *Social Science Computer Review.*, vol. 9, pp. 62–72, 1991.

[53] Tariq, M. B., Bhandakar, K., Valancius, V., Feamster, N., and Ammar, M., "Answering "what-if" deployment and configuration questions with wise: Techniques and deployment experience," tech. rep., Georgia Tech School

of Computer Science, Jan. 2010. `http://www.cc.gatech.edu/~mtariq/pub/wise-tr.pdf` (Date accessed: Jan. 2010).

[54] Tariq, M. B., Zeitoun, A., Valancius, V., Feamster, N., and Ammar, M., "Answering "What-if" Deployment and Configuration Questions with WISE," in *Proc. ACM SIGCOMM*, (Seattle, WA), Aug. 2008.

[55] Wasserman, L., *All of Statistics: A Concise Course in Statistical Inference.* Springer, 2003.

[56] Wolberg, J. R., *Data Analysis Using The Method Of Least Squares: Extracting The Most Information From Experiments.* Springer, 2005.

[57] Zhang, Y., Mao, Z. M., and Zhang, M., "Ascertaining the Reality of Network Neutrality Violation in Backbone ISPs," in *Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII)*, (Calgary, Alberta. Canada.), Oct. 2008.