Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Telecommunication Engineering

MULTIDIMENSIONAL CLOUD LATENCY MONITORING AND APPLICATIONS

Doctoral Thesis

Ing. Ondřej Tománek, M.B.A.

Ph.D. programme: P2612 Electrical Engineering and Information Technology
Branch of study: 2601V013 Telecommunication Engineering
Supervisor: Dr. Lukáš Kencl

Prague, November 2018

I hereby declare that I elaborated this doctoral thesis independently and used only sources in the bibliography.

Abstract

This thesis discusses a novel concept called *Multidimensional Cloud network latency monitoring* and applications of collected measurements.

Network latency has a direct impact on overall service performance, and, consequently, on the end–user experience. Unfortunately, latency is generally difficult to model, predict or control. Despite that, we show that a lot of information about Cloud service latency can be derived via thoughtfully designed active probing and, in turn, provide a foundation for improving various aspects of Cloud service performance.

We initially propose a novel distributed monitoring methodology, based on measuring Cloud service at multiple dimensions it is composed of. This approach is cost effective, has the global reach, does not perturb the service and provides dependable measurements. We then offer a lightweight planet–scale implementation that serves as our data collection platform and confirms the feasibility of our approach.

Later on, we leverage measurements, collected across two major public Cloud service providers, to improve Cloud services via three data–driven applications. Firstly, we use a set of descriptive–statistic methods and timeseries analyses to evaluate Cloud service, revealing both the known and unknown anomalous behaviors, as well as various insights and evolutionary trends. Secondly, we utilize preprocessed measurements for performance modeling of Cloud services, which, through a subsequent benchmarking process, are shown to exhibit certain unexpected and undesired spatio–temporal performance variations at different infrastructure levels and scales. Lastly, we show how these performance variations can be exploited by an informed gateway middleware to minimize adverse network effects and avoid many issues on the Cloud end.

Applicability of the multidimensional monitoring is confirmed by, but not limited to the presented application areas. Nascent service models, computing paradigms and socio–economic trends present tremendous monitoring opportunities in the future.

Index terms: Cloud; Latency; Monitoring; Profiling; Benchmarking; Optimization

Abstrakt

Předmětem této disertační práce je nový koncept *Vícerozměrného monitorování síťové latence Cloud Computingu* a aplikace naměřených dat.

Síťová latence má přímý dopad jak na výkonnost služeb, tak i na následnou uživatelskou zkušenost. Obecně však není jednoduché síťovou latenci modelovat, predikovat ani kontrolovat. Přesto v této práci ukazujeme, že pečlivě navržené aktivní měření latence poskytuje detailní vhled do výkonnosti Cloudových služeb a tím staví základy pro zlepšování různých jejich aspektů.

V této práci nejprve navrhujeme novou metodologii distribuovaného monitorování latence, založené na současném měření dimenzí, ze kterých se Cloudová služba skládá. Takový přístup je cenově dostupný, má globální rozsah, nenarušuje službu a poskytuje spolehlivá data. Vhodnost metodologie následně potvrzujeme návrhem a implementací metody sběru dat, založené na globální měřící platformě.

V práci dále pomocí vytvořených technik dlouhodobě monitorujeme dva hlavní Cloudové poskytovatele a naměřená data využíváme ke zlepšení služeb pomocí třech nových, na datech o latenci založených, aplikací. Nejprve využíváme metod popisné statistiky a analýzy časových řad ke zhodnocení Cloudových služeb, odkrývání známých i neznámých anomálií, vhledů do chování služeb a jejich vývojových trendů. Následně využíváme předzpracovaná data k modelování výkonnosti Cloudových služeb a jejich benchmarkingu, odhalujícím neočekávané nežádoucí rozdíly ve výkonnosti na různých vrstvách a granularitách infrastruktury. Na závěr ukazujeme, jak lze rozdíly ve výkonnosti služeb využít chytrou domácí branou k minimalizaci negativních síťových vlivů a vyhnutí se velkému množství problémů na straně poskytovatele Cloudových služeb.

Využitelnost vícerozměrného monitorování je uvedenými aplikačními oblastmi potvrzena, nikoliv však omezena. Vznikající modely služeb, výpočetní paradigmata a socioekonomické trendy nabízí velkou využitelnost monitorování i v budoucnosti.

Klíčová slova: Cloud; Latence; Monitorování; Profilování; Benchmarking; Optimalizace

To every single bit of my family.

Acknowledgments

Firstly, it is a great pleasure to express my gratitude to my advisor, Dr. Lukáš Kencl, for all the support en route to this thesis, the wisdom he shared, roles he allowed me to impersonate and, above all, always making me dare for more.

Special thanks go to the former RDC lab members, Jan Staněk and Michal Ficek, who made the place both inspiring and aspiring. I am deeply indebted to my Cisco mentors and colleagues for their patience, support and for serving as role models. Also, many thanks to the fellow Cisco interns for the wonderful Bay Area memories.

I much appreciate my co–authors, Pavol Mulinka, Zdeňek Kouba and Vojtěch Uhlíř, for great teamwork, much beyond putting manuscripts together. Also, my sincere thanks go to the anonymous reviewers, whose constructive comments and suggestions improved my research including this thesis.

I am grateful for the great place to conduct academic research, provided by the Czech Technical University in Prague and Department of Telecommunication Engineering. Special acknowledgments go to the funding sources and industry partners, for providing foundation for the practical research.

Last, but certainly not least, my heartiest thanks go to my mum, dad, brother and my fiancée, Aneta, for always being there for me.

That which is monitored improves.

– R. JAIN

Contents

List of Figures

List of Tables

List of Acronyms

ACK	TCP Acknowledge Flag
API	Application Programming Interface
AI	Artificial Intelligence
AR	Augmented Reality
ARIMA	Autoregressive Integrated Moving Average
AWS	Amazon Web Services
b	bit
B	Byte
BGP	Border Gateway Protocol
CBWFQ	Class–Based Weighted Fair Queueing
CIB	Collective Intelligence Benchmarking
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CLAudit	Cloud Latency Auditing Platform
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSP	Cloud Service Provider
CSV	Comma–Separated Values
CTU	Czech Technical University in Prague
CV	Coefficient of Variation
CX	Customer Experience
DC	Data Center
DNS	Domain Name System

EC2	Elastic Compute Cloud
ERP	Enterprise Resource Planning
GC	Garbage Collector
GCD	Great Circle Distance
GIS	Geographic Information System
GNU	GNU's Not Unix
HAN	Home Area Network
HCA	Hierarchical Cluster Analysis
HDR	High Dynamic Range
HDTV	High–Definition Television
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IATA	International Air Transport Association
IBM	International Business Machines Corporation
ICMP	Internet Control Message Protocol
i.i.d.	independent and identically distributed
I/O	Input / Output
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
ISP	Internet Service Provider
IXP	Internet Exchange Point
KPI	Key Performance Indicator
LAN	Local Area Network
LLQ	Low–Latency Queuing
LXC	Linux Containers
MAC	Media Access Control
MATLAB	Matrix Laboratory
MEC	Mobile Edge Computing

MILP Mixed–Integer Linear Program

ML Machine Learning

MPI Message Passing Interface

MRTG Multi Router Traffic Grapher

MQTT Message Queuing Telemetry Transport

NAT Network Address Translation

NREN National Research and Education Network

NTP Network Time Protocol

ODC–By Open Data Commons Attribution License

Ops Operations

OSI Open Systems Interconnection

OSS Open–Source Software

OS Operating System

OTA Over the Air

PaaS Platform as a Service

PCA Principal Component Analysis

PDF Probability Density Function

PDO PHP Data Objects

PHP PHP: Hypertext Preprocessor

PON Passive Optical Network

PoP Point of Presence

PSD Power Spectral Density

RAID Redundant Array of Independent Disks

RDC Research and Development Centre

RDMA Remote Direct Memory Access

RDMS Relational Database Management System

RDS Relational Database Service

RFC Request for Comments

RIPE Réseaux IP Européens

RMON Remote Network Monitoring

RTT Round Trip Time

S3 Simple Storage Service

SaaS Software as a Service

SAN Storage Area Network

SCP Secure Copy Protocol

SDN Software Defined Networking

SIEM Security Information and Event Management

SLA Service–Level Agreement

SNMP Simple Network Management Protocol

SOHO Small Office Home Office

SQL Structured Query Language

SSH Secure Shell

SYN TCP Synchronize Flag

TCO Total Cost of Ownership

TCP Transmission Control Protocol

TPC Transaction Processing Performance Council

ToS Type of Service

TTL Time to Live

UDP User Datagram Protocol

UI User Interface

URL Uniform Resource Locator

URLLC Ultra–Reliable Low–Latency Communication

UTC Coordinated Universal Time

VoIP Voice over IP

VP Vantage Point

VM Virtual Machine

VR Virtual Reality

WAN Wide Area Network

WGS World Geodetic System

WWW World Wide Web

YAML YAML Ain't Markup Language

QoS Quality of Service

Chapter 1

Introduction

1.1 Cloud Latency Monitoring

Time is among human's most valuable resources. With our lives becoming increasingly more digital, much of a lifetime is spent waiting for issued digital tasks (application launch, call routing, search queries, transaction processing, authentication, tactile control etc.) to complete. At the heart of these tasks are compute, network and storage operations – each taking certain time to finish. That time is what we call *latency*.

Network plays a vital role every time when a non–local operation takes place. Network–incurred latency includes signal propagation, middlebox processing, routing, switching, policing and various other decision–making to ensure a communication that respects interests of all involved parties. Under an increasingly popular Cloud Computing service model, controlling latency is difficult because of traits such as virtualization, offload, multitenancy and complex distributed service architectures. A certain portion of human lifetime is thus left at a mercy of a technology that is controlled and predictable to only some respect. We aim to better understand and decrease amount of time people and machines lose when using networked systems.

The first problem we address is **how to rigorously monitor network latency under the Cloud Computing service model**. The second problem we address is **how to leverage these measurements to improve Cloud services**. Therefore, the work in this thesis concentrates on a Cloud Computing latency monitoring and applications of measurements thereof – all for the benefit of user experience and time saving.

The key contributions of this work are a *suitable methodology of Cloud Computing latency monitoring* and three *data–driven methodologies for improved network profiling, benchmarking and connectivity to Cloud services.*

There is no single metric that captures latency well. An ideal latency monitoring output would describe complete distributions of latencies at every place of Cloud Computing networks. There is, however, no feasible way of obtaining that. Hence, our goal is to strike the right balance between deployment cost, measurement reach, sampling frequency and data dependability. There are many direct and secondary advantages of deploying such a global distributed *latency monitoring* solution:

- Holistic unbiased view of end–to–end Cloud performance from a 3$^{\text{rd}}$–party viewpoint;
- Sensing platform that reveals coincidences and attributes failures and various other events at local, regional or global scopes;
- Manual detection and training data for automized detection of Cloud events (issues, improvements, trends etc.) together with their origin;
- Decision support for infrastructure changes, application migration or outsourcing;
- Evidence for Service–Level Agreement (SLA) accountability;
- Finding the frequently–used network segments and tuning system parameters to optimize performance;
- Measuring resource utilization and finding performance bottlenecks;
- Characterizing workload for capacity planning and for creating test workloads;
- Finding model parameters to validate models and to develop inputs for models.

The gists of our three respective applications, built on top of the monitoring, are as follows:

Cloud network profiling application is based on descriptive–statistic metrics that manually profile latency timeseries and spatio–temporarily look for coincidences of events therein across protocol layers of the network communication stack, vantage point locations and Cloud resources.

Longitudinal Cloud benchmarking application is based on data transformations and a notion of distance in n–dimensional vector spaces that characterize performance of Cloud resources being compared.

Cloud connectivity optimization application is based on Convex Optimization, specifically binary Mixed–Integer Linear Program (MILP) striving to minimize adverse network effects on network traffic.

We demonstrate applicability of the three applications using *case studies of actual ground–truth latency* data of the major Cloud service providers – Microsoft Azure and Amazon Web Services (AWS).

We used to provide *near–realtime data* and still provide *archived data* in a form of open datasets on the project website `http://claudit.feld.cvut.cz`

1.2 Outline

In Chapter 2, we present essential background knowledge of network latency and, specifically, Cloud Computing network latency.

In Chapter 3, we review the Cloud monitoring state–of–the–art, as well as state–of–the–art of its discussed application areas – Cloud network profiling, Cloud service benchmarking and Cloud connectivity optimization.

In Chapter 4, we describe in detail the key underlying innovation – the *Multidimensional Cloud latency monitoring* methodology, together with its properties. We also present a performance evaluation, yielding suitable values for monitoring parameters.

In Chapter 5, we devise a platform for multidimensional–measurements capturing and present its implementation and deployment at the global scale.

In Chapter 6, we present and evaluate performance of the three novel applications of the collected multidimensional Cloud latency measurements for Cloud service improvements – *a methodology for revealing Cloud service insights and profiling Cloud network behavior*, a *Cloud service benchmarking methodology* and a *Cloud–connectivity optimization methodology.*

Finally, in Chapter 7, we give concluding remarks on the future applicability of the methodology and mention related open issues.

1.3 Contributions

- A new methodology of Cloud network latency monitoring is presented. It is based on continuous large–scale active probing of multiple Cloud resources at multiple network protocol layers from a global network or Internet vantage points. It is lightweight, easy–to–deploy, dependable and stores little information;
- A method for multidimensional measurements capture was devised and implemented using PlanetLab and basic–tier public Cloud provider resources;
- Actual real–world measurements were continuously collected using Cloud monitoring platform, visualized in a near–realtime, archived online and are still provided in the form of a publicly available dataset;
- A mutidimensional–measurement timeseries analyses are presented, together with the revealed insights, anomalous behaviors and performance trends of Cloud services;
- A new longitudinal methodology for latency–based Cloud service benchmarking is presented that accurately reflects application requirements by leveraging preprocessed multidimensional measurements;
- A new MILP–based methodology for Cloud connectivity optimization is presented. It minimizes cumulative adverse network effects on Cloud–bound traffic and avoids problematic destinations by using informed–gateway decisions. A naïve implementation using router application plane is offered;
- A performance evaluation of the multidimensional latency monitoring is carried out, as well as performance evaluations of the respective methodologies of the data–driven applications;
- Large–scale evaluation of Cloud service latency of two major Cloud service providers, Microsoft Azure and Amazon AWS, is carried out through case studies that use aforementioned methods. The results reveal notable hidden trends, events, infrastructure changes, performance differences and optimization potential at granularities of Cloud service providers, datacenters and resources;
- Prototypes of all methods were implemented in MATLAB or Python.

Chapter 2

Latency Background

In this Chapter, we present essential background knowledge and properties of communication network latency. State–of–the–art techniques that tackle latency–related problems are also presented. We then describe how the complex Cloud Computing architectures change the behavior of network latency and how research industries and academia react by adjusting latency research works.

2.1 Network Latency

Computer networks necessarily constrain throughput, introduce latencies between end systems and can actually lose packets [76, 102]. Network latencies a packet suffers from in intermediate nodes include processing, queuing, transmission and propagation latency. Latencies a packet suffers from in end–systems include packetization, response composing and shared–media latencies. *End–to–end latency* is then a sum of all these blocking latencies along the entire communication path. Figure 2.1 depicts the first category – the nodal latencies, which are most commonly modeled as the Sum 2.1.

$$q + v(t) \tag{2.1}$$

The sum has two components, q is a constant, and $v(t)$ is a random process with a long–tailed, usually unknown distribution (e.g., Exponential or Gamma). The latency of each packet is typically assumed to be independent and identically distributed (i.i.d.). Previous work has shown that latency values are not truly i.i.d. for two main reasons. First, the bursty behavior of network latency [42], and second, network latencies have a

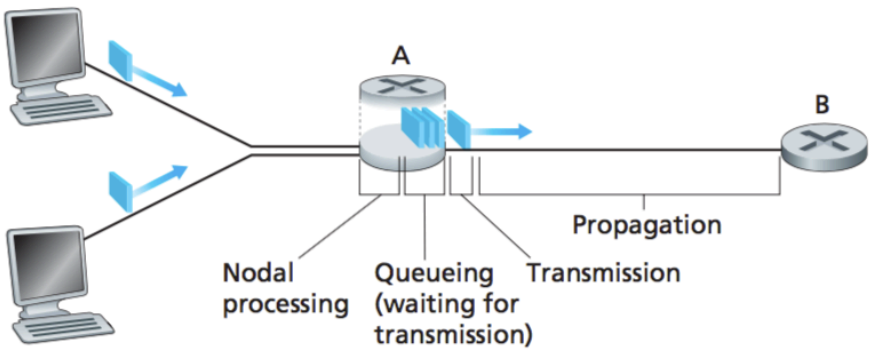


Figure 2.1: ***Nodal latencies at router*** *[76].*

periodic behavior [98] that is affected by the time–of–day effects, corresponding to the behavior of human end–users [97].

A time interval between issuing a request and receiving a response back is called a *response time*. In the case when both the request and the response are composed of a single packet, response time often becomes similar to end–to–end latency and a single round trip takes place. It is important to draw a distinction between *service time* and *response time*, as the latter grows when arrival rate is higher than service rate. Figure 2.2 shows how a system response time changes with an increasing load or decreasing resource availability (e.g., running out of queue capacities, overloading nodal CPUs or congesting shared medium).

Inevitably, latency occasionally gets out of reasonable bounds – a phenomenon called *tail latency*. An extent to which it is allowed to do so is captured by Service–Level Agreements (SLAs), which, among others, prescribe maximum RTT values to latency percentiles. Although a single occurrence of tail latency is by definition rare, its encounter with end user is not. For example, a dynamic nature of WWW implies webpages that are composed of many resources, often non–local (Table 2.1). The third column shows a likelihood of one access experiencing the 99^{th} percentile. A typical web user's session involves five page loads, averaging 40 resources per page, which means just 0.003% users will not experience something worse than 95^{th} percentile. 18% of users are going to experience response time worse than 99.9^{th} percentile, less than 5% response time worse than 99.97^{th} percentile and less than 1% response time worse than 99.995^{th} percentile. Thus, tail latency is what most web user will see.

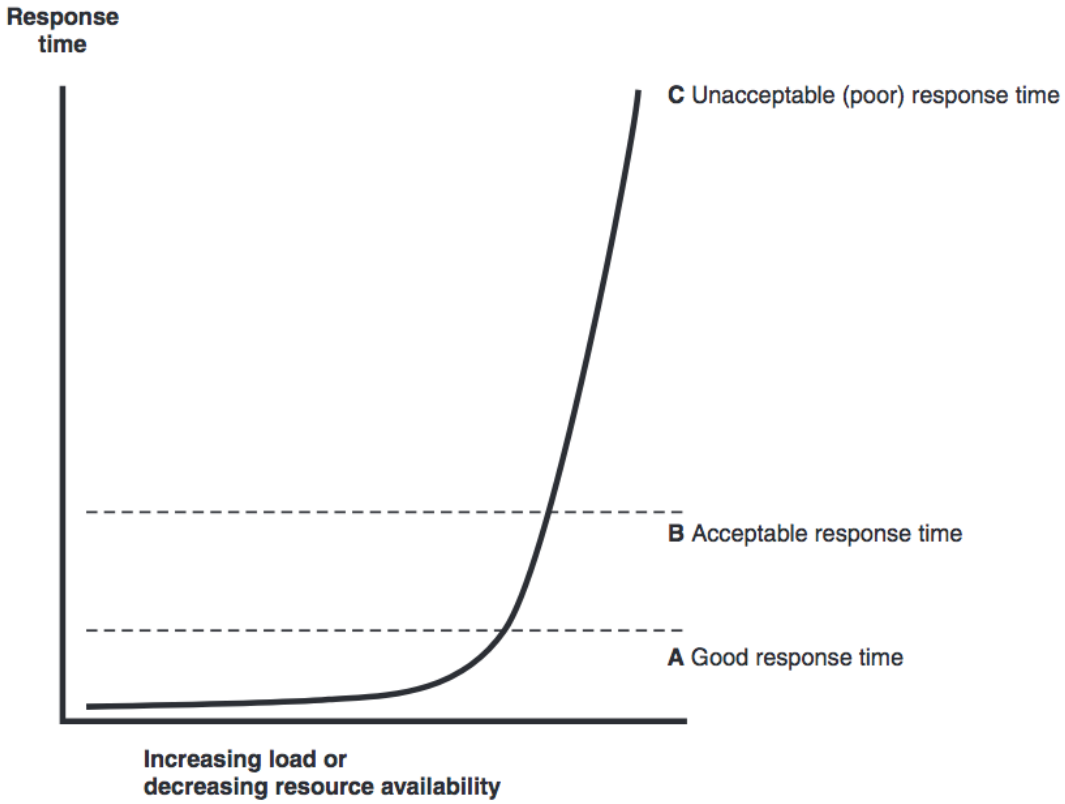


Figure 2.2: ***Response time spectrum.*** *Effect of response time in the face of increasing load, going beyond the system's operating range [9]*

Table 2.1: ***Fraction of web users experiencing 99–th latency percentile or worse*** *[116].*

Website	# of subordinate requests	Page loads experiencing 99%'ile or worse
amazon.com	190	85.2%
kohls.com	204	87.1%
jcrew.com	112	67.6%
saksfifthavenue.com	109	66.5%
nytimes.com	173	82.4%
cnn.com	279	93.9%
twitter.com	87	58.3%
pinterest.com	84	57%
facebook.com	178	83.3%
google.com	31	26.7%
google.com/search?q=latency	76	53.4%

Most SLA monitoring systems stop at the 99^{th} percentile, for practical reasons. The data collected by most monitoring systems is usually summarized in small, 5 to 10 second windows. Thus, measurement fidelity is lost, as deriving five nines for a minute or hour from a set of small samples of percentiles is not possible. A quick feasible practical approach to latency evaluation is to define SLAs and plot their requirements. Then, different production–application scenarios using different configurations and different workloads should be run. The results indicate whether SLAs are met, or how many machines need to be provisioned to do so (Figure 2.3). For CSPs, SLAs are important for estimating insurance costs.

Many tools exist for storing latency, such as *HDRHistogram* [18]. This particular tool supports recording and analyzing sampled data value counts across a configurable integer value range, with configurable value precision within the range. Value precision is expressed as the number of significant digits in the value record, and provides control over value quantization behavior across the value range and the subsequent value resolution at any given level. Distribution can be reported using percentiles, linear or logarithmic value buckets, or mean and standard deviation. HDRHistogram can handle high–volume production data (Figure 2.3).

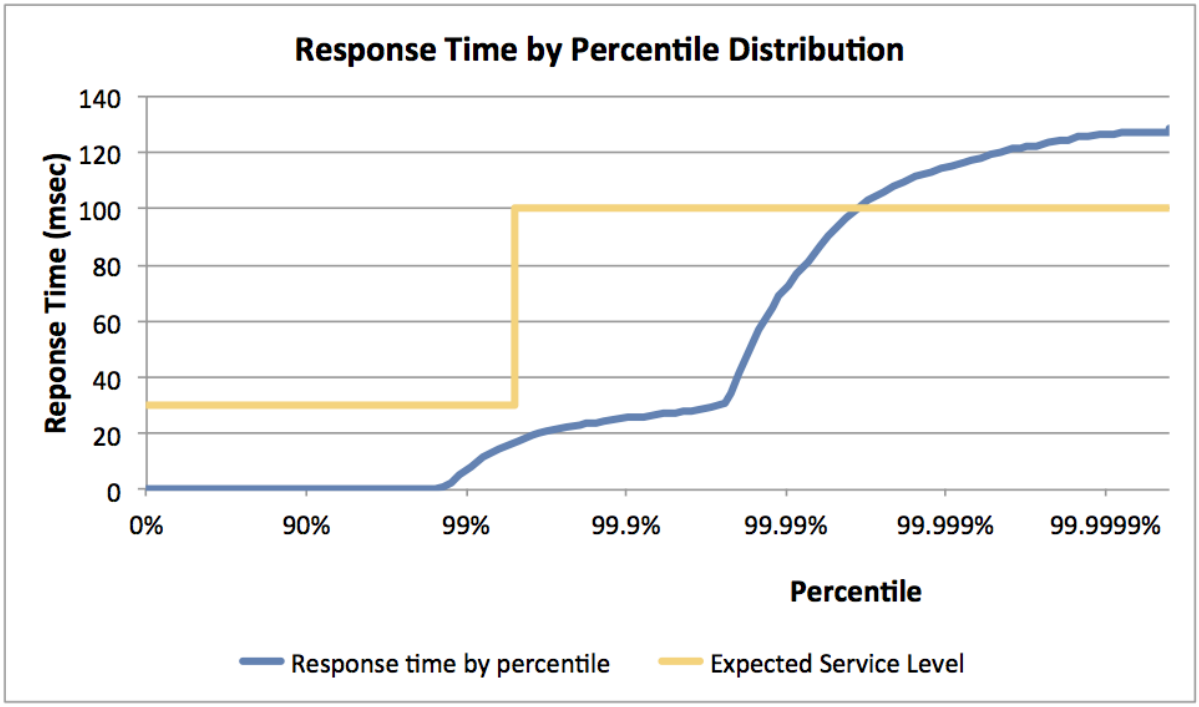


Figure 2.3: ***SLA versus actual response time captured by HDRHistogram tool*** *[18].*

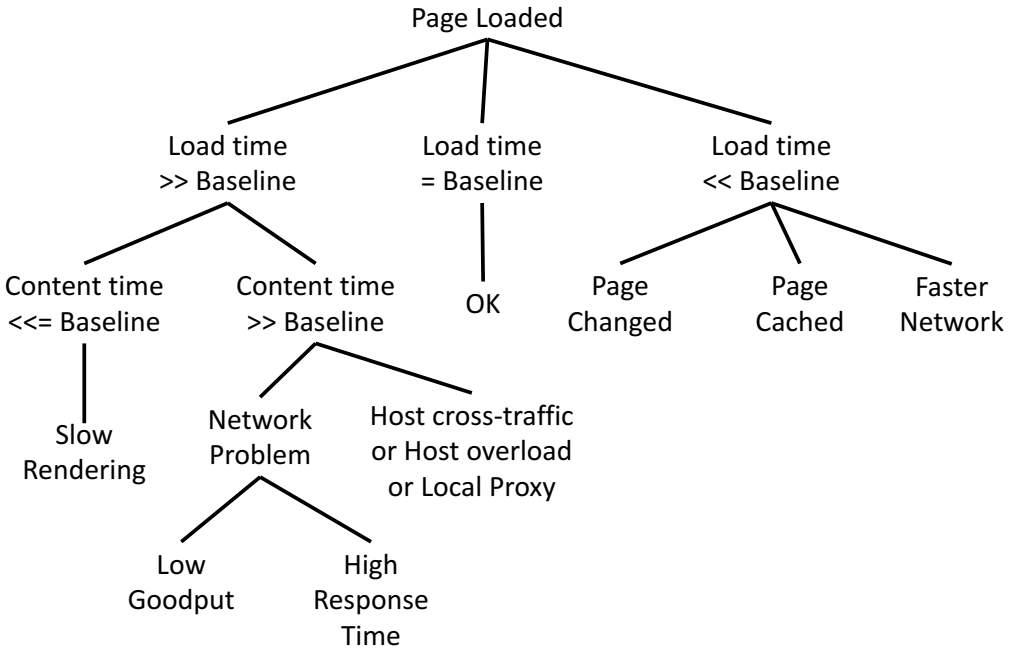


Figure 2.4: ***Diagnosing web browsing issues using latencies [59].***

Latency does have many other uses, such as reasoning about issues and failures of web browsing. Fine–grained partitioning of the end–to–end latency into respective latencies of computation, network and storage operations can aid in diagnosing and resolving user complaints. Figure 2.4 shows diagnosis logic when webpage loads successfully.

With evolving technologies clearly improving queuing, processing and transmission latency [135], understanding propagation latency becomes even more important. The propagation latency is, above all, dictated by the *Speed of light barrier* and becomes a dominant contributor to the end–to–end latency when communication distance increases. Propagation latency thus explains, why Wide Area Network (WAN) latencies are an order of magnitude higher than Local Area Network (LAN) latencies. Distances between communication endpoints can be conveniently approximated by *Great Circle Distance (GCD)* – the shortest distance between two points on the WGS84 ellipsoid calculated using *geod* library, measured along the surface and ignoring differences in elevation [62]. Section 6.2 shows examples of the lower propagation–latency bounds over various GCDs (see Table 6.3). The *actual* data path is often different from the path suggested by the GCD. Propagation speed between communication endpoints also depends on the physical medium of the links (e.g., fiber optics, twisted–pair copper wire, wireless) and is in the range of $2 \cdot 10^8$ m/s to $3 \cdot 10^8$ m/s. Using a distance between communication endpoints and the theoretical maximum propagation speed, one can calculate the lower bound that constrains minimum

achievable latency (Set membership 2.2). Latency, once added, never goes away.

$$\mathrm{RTT}_{\min}(\mathrm{gcd}(\mathrm{src},\ \mathrm{dest})) \in \Omega\Big(\frac{2.\mathrm{gcd}(\mathrm{src},\ \mathrm{dest})}{c_{\mathrm{fiber}}}\Big) \tag{2.2}$$

Ensuring lowest possible latency is not always practical, given high monetary costs and effort involved. Several classes of network applications have little or no sensitivity to latency and related quality parameters and, thus live just fine with unstable high-latency lossy environment [51]. However, in the face of trends like network convergence and immersive technology, ensuring a well–performing network becomes ever more important. Figure 2.5 shows latency sensitivity and a corresponding maximum communication distance of selected traditional applications. Requirements in the form of SLAs should reflect various demands that applications have on communication networks. Proper network measurements and monitoring should provide data to evaluate these requirements.

2.1.1 Related Work

The majority of the past latency comprehension and tackling efforts have come from the Internet *Traffic engineering* domain, striving to ensure sufficient Quality of Service (QoS). Results exist in the form of theoretical concepts, industry solutions, RFCs

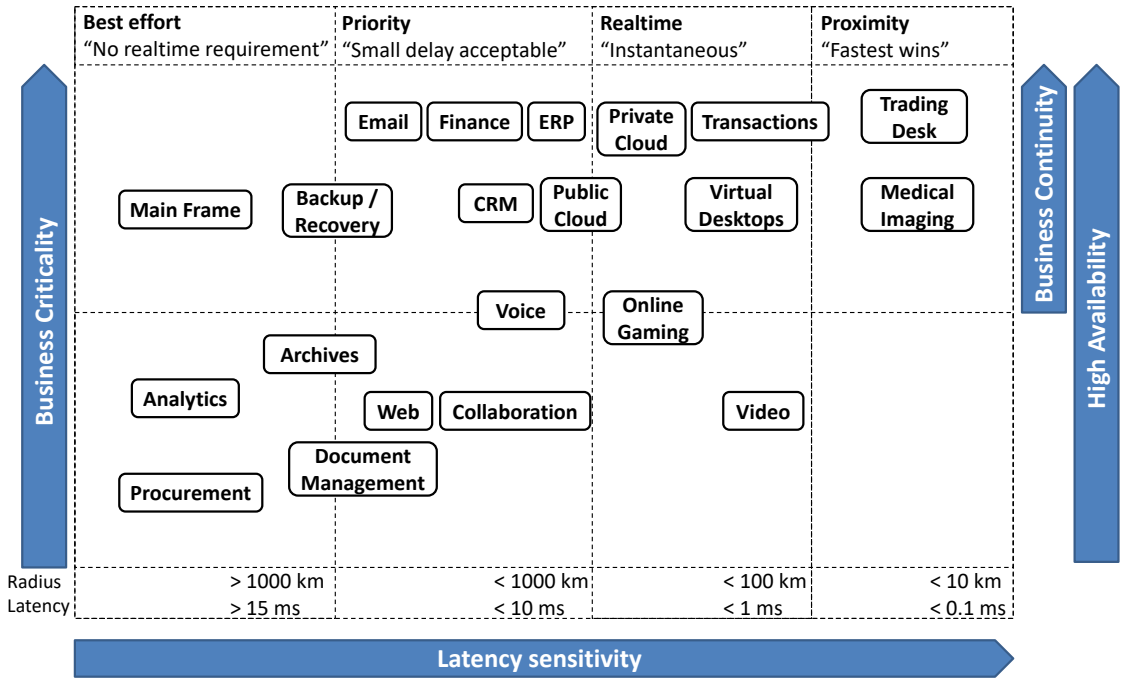


Figure 2.5: ***Latency sensitivity and corresponding maximum communication distance of selected traditional applications*** *[114].*

and standards. Theoretical foundation of latency engineering was given by *Queuing theory* [70, 81] and *Network calculus* [82, 71]. Specific solutions such as *Guaranteed Service networks* or end–to–end jitter bounds were discussed in many works [45, 58, 80]. Because of the specificity and limited applicability of these solutions to today's computer netwoks, overprovisioning [94] remains the most popular way of achieving QoS.

Many negative latency–related observations have been published, e.g., that significant portion of Internet traffic suffers from routing pathologies or that variations in end–to–end latency indicate long congestion periods [105, 104, 106]. More bad news include latency unpredictability [74, 37] and the deep–seated tail latency [132, 87, 57, 60].

2.2 Cloud Computing Network Latency

By *Cloud Computing* we understand a model for enabling ubiquitous, on–demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service–provider interaction [93]. Cloud, or a Cloud Computing system, is a set of elements and logical components participating in a particular Cloud Computing scenario (see example in Figure 2.6). By *Cloud service* we understand a set of Cloud *service–provider* offerings to *tenants*. By *Cloud application* we understand tenant's application deployed on top of selected Cloud services, often serving *end users* via client appliances. *Cloud service providers* (CSPs) primarily provide services using *datacenters* (DCs), which host computation, storage and network *resources*.

Internet growth and Cloud Computing service model pronounce the long–standing problems related to network latency. Among the top causes are a number of subordinate sessions, scale and great demands of the new applications, both for computing resources and for quality of the communication networks. While the more obvious problem of computing–resource allocation has been deeply investigated, research in improving network–quality parameters, except throughput, has not got as much early attention [52]. As a consequence, end–user's time and patience are often the price, and deployment of latency–sensitive applications in the public Cloud has been slow, often favoring proprietary solutions. Cloud customers – end users and tenants – thus still wait for satisfactory answers to their application–performance requirements.

Cloud–service latency plays a vital role in many kinds of applications, from the

simple web–based services to the tactile–controlled distributed games. There are orders–of–magnitude of difference between their latency requirements, ranging from units of seconds to units of milliseconds. Low–latency demands are not limited to those applications, but also to the properties inherent to the Cloud–Computing infrastructure, such as replication, task distribution, sharing, synchronization, offload or rapid scaling. Operation with stringent latency requirements is thus possible only after extensive latency–based optimizations, as there is no easy scale–out strategy.

Expressing latency using analytic methods slowly becomes next–to–impossible, because of diverse interdependent traffic patterns and complex use cases [129]. Due to the generally local nature of latency–reduction tools, global end–to–end optimization, that takes all current Cloud state information into account, might be neither effective, nor feasible, without a global informed view. Promising end–to–end efforts (e.g., DiffServ QoS) were either not deployed at large scale or remained only in paper form. Rigorous monitoring and understanding latency origins and symptoms are thus prerequisites to approaching the Cloud latency problem at the global scale.

It is not just the *distributed computing nature* that changes latency behavior in Cloud Computing era. Equipment *virtualization* and *Virtual Machine (VM)* migration are techniques for maximizing resource utilization. They, however, impose certain network demands and add an extra software layer that results in extra system overhead.

Cloud Computing also makes heavy use of *offload technologies* like CDNs or

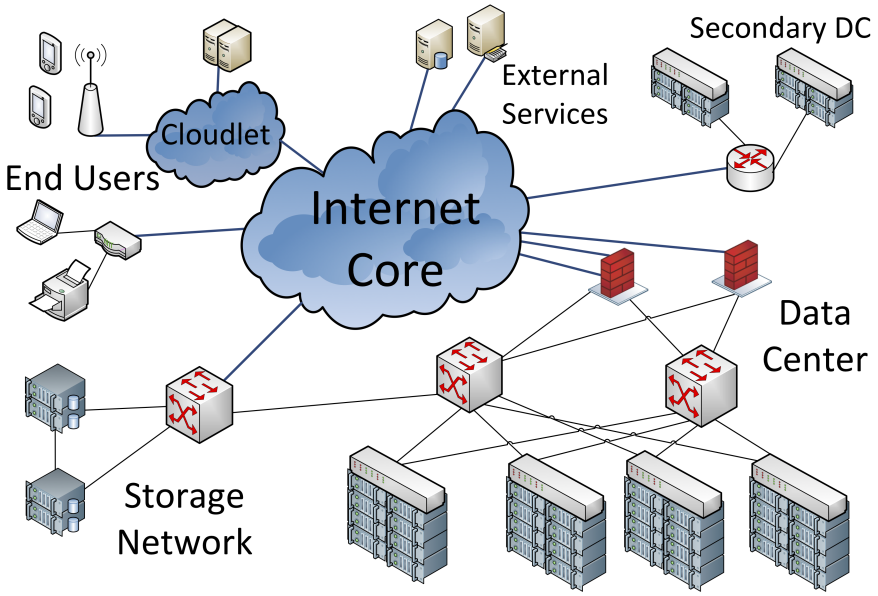
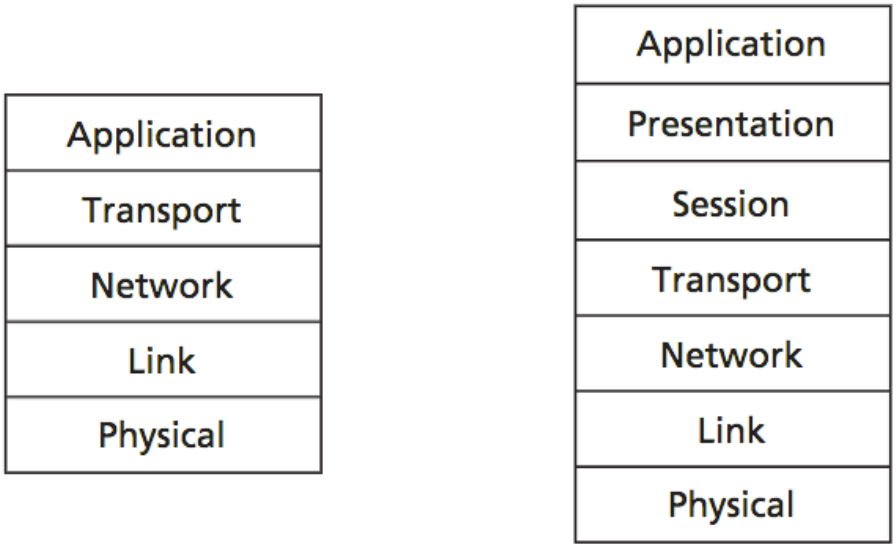


Figure 2.6: ***Cloud Computing system.*** *Example of various elements and components involved in Cloud Computing. Elements and components can also be viewed as latency–contributing sources.*



(a) *Five–layer Internet protocol stack* **(b)** *Seven–layer ISO/OSI model*

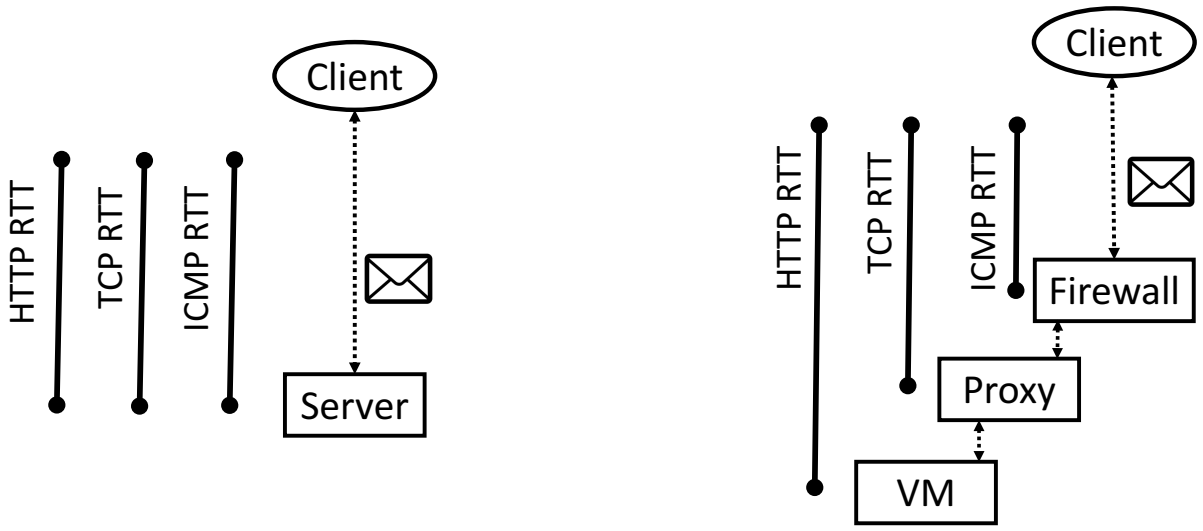
Figure 2.7: ***The Internet protocol stack (a) and ISO/OSI reference model (b).***

Cloudlets, which may silently reduce latency by conducting a computation or serving content from a location between the end user and the provider Mega–DC.

Lastly, in Cloud Computing scenarios, *protocol termination* of layers of network communication stack (Figure 2.7) is often blurry. In conventional client–server interaction, the server terminates all the upper protocol layers (i.e., responds to ICMP requests, TCP handshakes and HTTP queries). In a typical Cloud Computing scenario, however, ICMP request is terminated or answered by edge firewall; TCP handshake is executed with a TCP proxy or a WAN accelerator; and HTTP query is answered by a web cache or VM. See Figure 2.8 for illustration. Worse still, adding to the ICMP's slow–path nature are filtering, rate limiting and depreferential services that blur the RTT measurements or even prevent the round trip from happening.

All aforementioned Cloud properties complicate correspondence of latency to distance and prevent the conventional Internet WAN measurement techniques to be easily adjusted for general Cloud Computing needs. There was some success in modeling latency of specific environments (Cloudlet–to–Cloud and cellular–to–Cloud latency can be roughly approximated using PDF in Equation 2.3, which denotes the *Rayleigh distribution* [77]).

$$f(x,\sigma) = \frac{x}{\sigma^2} e^{x^2/2\sigma^2}, \quad x \geq 0 \tag{2.3}$$



(a) *Traditional RTTs* **(b)** *Cloud RTTs*

Figure 2.8: ***Cloud versus traditional RTTs of various protocols.*** *Similar round–trip–times across protocol layers are observed in a conventional client–server scenario (a). Significant differences are observed in a Cloud Computing scenario (b), where different protocol layers are terminated at different middleboxes and sometimes even at different locations.*

2.2.1 Related Work

Latency of specific Cloud applications and aspects was examined [50, 103], often using tools designed in academia (such as *Fathom* [59] or *Flowping* [127]). Many works discuss user experience of Cloud performance [91, 96]. Given an often unknown Cloud topology and technology, network latency tomography [123] and latency predictions [89, 66, 129] remain as important techniques.

Applicable to the Cloud are industry and research techniques for reducing Internet WAN latency, either by acceleration heuristics [112] or informed BGP path–selection algorithms [131]. CDN selection algorithms [78] with redundant requests [65, 128] can be used to remove the Cloud tail latency. Industry and academia have converged towards improving performance by moving the Cloud closer to the end–user and offloading strategic responsibilities and computation from remote Mega DCs based on latency. Concepts such as Cloudlets, MEC or Fog Computing are successfully being deployed [111, 126].

Innovations inside the DC include data path management [107, 54, 40], transport optimization, adjusting TCP behavior within the DC network or proposing entirely new protocols (*DCTCP* [39], *D3* [130], *D2TCP* [125], *PDQ* [67], *pFabric* [41] or *delay–based TCP* [83]). Entire research groups exist that focus on predictable DC performance [44].

Chapter 3

State of the Art

In this Chapter, we are going to review the previous fundamental results in fields discussed in this thesis. Monitoring of Cloud Computing network performance has recently been studied in Computer science and Networking industry and a wide array of works exists in this area. We present those approaches that are seminal or share common traits and objectives with our approach.

In the later Sections of this Chapter, we examine previous works in three applications areas for which we present data–driven methodologies built on top of monitoring data. These areas are Cloud network profiling, Cloud–service benchmarking and Cloud–service connectivity optimization.

3.1 Cloud Latency Monitoring

Cloud latency monitor is a tool used to observe latency activities on a Cloud system. To an outside observer, a remote Cloud DC appears as a blackbox and various tricks are needed to separate various possible latency causes, e.g., by carefully planning the architecture and deployment of Cloud monitoring.

Very limited insights can be obtained using a commonplace *one–dimensional monitoring* approach, i.e., deploying a single Vantage Point (VP) to periodically probe an arbitrary Cloud target via a single protocol. Unnoticed go, for example, many causes of network failures or particular path segments affecting the Cloud–service performance. The main reasons for one–dimensional monitoring's prevalence include limited global presence of many tenants and monetary costs related to deploying additional VPs.

One–dimensional latency measurements are provided by free 3$^{\text{rd}}$–party services

like *CloudPing* [11] and *CloudWatch* [13], by network monitoring tools like *Cacti* [8], *Zabbix* [36], *MRTG* [24] and *Nagios* [25], by CSP–offered monitoring products like *Amazon CloudWatch* [1] and *Azure Monitor* [6], or by CSP–service dashboards like *AWS Service Health Dashboard* [3] and *Azure Status* [7]. CloudPing conducts a simple `httping` [19] test against several selected AWS DCs and reports mean RTT. CloudWatch also issues small HTTP requests, but instead of measuring only AWS frontend, it also measures services such as messaging and queuing buses. CSP dashboards and monitoring products are availability–oriented and do not report detailed latency numbers behind their health–checking probes.

Advanced commercially–provided monitoring tools (e.g., *Renesys* [29], *ThousandEyes* [34] and *Nyansa* [27]) generally do not publicly disclose important details of the measurement methodology and are usually not provided as a free 3$^{\text{rd}}$–party service. Many of the tools use only basic metrics, are network–administrator oriented, do not exhaustively leverage coincidences across all data dimensions and provide a limited automation of event interpretation, leaving much investigation to the tenant or the end user.

Renesys conducts global daily measurements from its operated `traceroute` infrastructure of DCs and virtual servers and captures metrics such as RTT and number of responding targets. Furthermore, results can be compared to the provided organizational or 3$^{\text{rd}}$–party datasets.

ThousandEyes installs agents on communication endpoints and captures metrics such as packet loss, DNS resolution time and response times across ThousandEyes' DCs and customer's agents. Nyansa works by deploying an entire sniffer inside a customer DC and uploading the sniffed traffic metadata to the Nyansa Cloud for processing–intensive analysis. User can then surf over results using a provided web interface. These on–premise deployment architectures raise many trust and privacy–related questions that represent main adoption drawbacks.

Network–equipment vendors like *Cisco* provide vast tooling for latency reduction and monitoring at every layer (Figure 3.1). Since the advent of Cloud Computing, the tooling has evolved to support new needs and monitoring has been offered as a subscription–based Cloud SaaS.

CloudSleuth [12] and *CloudHarmony* [10] monitor response time of public CSPs from a global network of VPs and provide measurements for free. CloudHarmony uses 2 tests (ICMP Echo and 8B–file HTTP download). Both tests use RIPE Atlas network

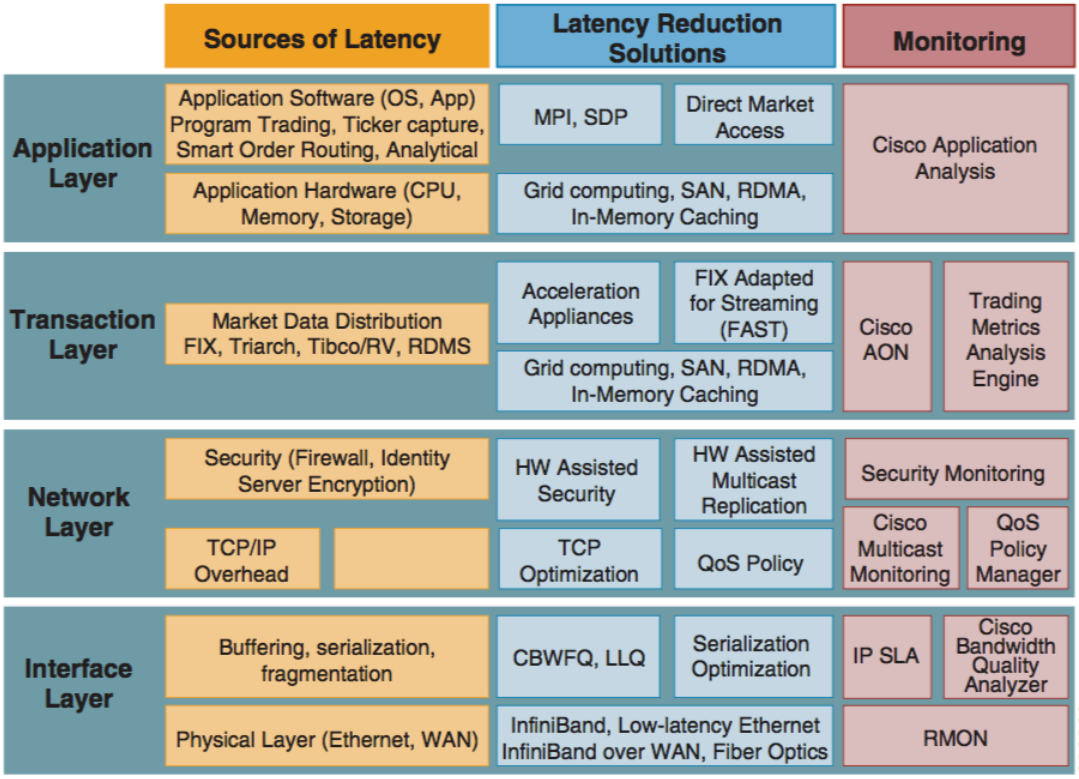


Figure 3.1: ***Layer–specific latency reduction and monitoring tooling from Cisco [55].***

testbed [30] to issue 12 sample requests during each test interval with the slowest 2 discarded, and a mean of the remaining 10 reported. CloudSleuth reports response time and availability of major IaaS and PaaS CSPs using web UI, shown in Figure 3.2. CloudSleuth works by hosting an identical sample application at DCs of measured CSPs. This application is continuously being queried by tens of Internet backbone VPs residing on several major continents.

Another approach is that of AIOps SIEM products like *Splunk* [32], which analyze, visualize and monitor machine–generated big data (e.g., log files or network traces). Advances insights are provided by correlating these voluminous data from various sources. Splunk ML toolkit allows to run user–supplied custom algorithms.

Wang, Huang, Li et al. [129] and Dhawan, Samuel, Teixeira et al. [59] have taken a browser–based measurement approach. The latter team has developed a platform called *Fathom*, implemented as a web–browser plugin that helps users diagnose their connection and provides researchers and website owners with performance statistics from the edge. Fathom's nature and implementation provide end–user perspective, portability, asynchronicity and millisecond accuracy. However, it requires users to install a piece of

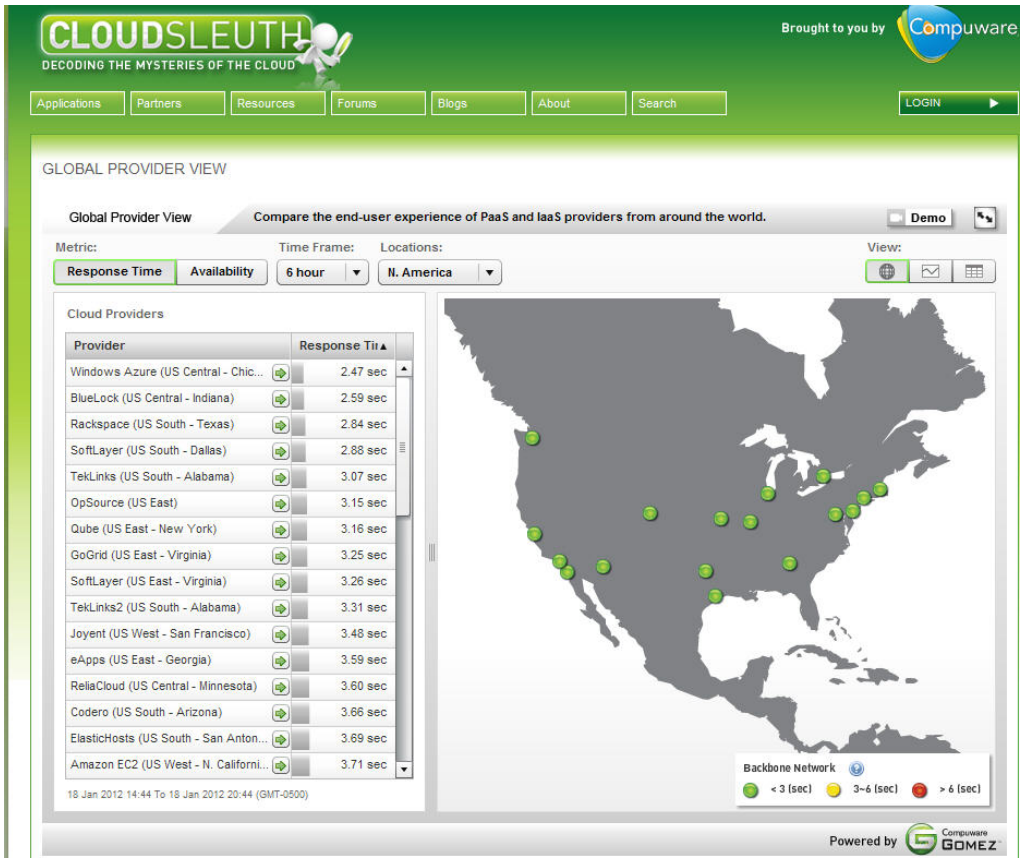


Figure 3.2: ***CloudSleuth CSP monitoring UI.***

software and, as such, depends on measured webpage being opened sufficiently long, is sensitive to stress or concurrent browsing conditions and slows page loads.

Li, Yang, Kandula et al. [86], as part of their *CloudCmp* test suite, measure intra–DC network, inter–DC network and WAN. They measure path latency using `ping` and TCP throughput using `iperf` to report performance of intra–DC and inter–DC networks. They use the optimal WAN latency (defined as the minimum latency between a hundreds of PlanetLab VPs and any DC owned by a CSP) to report WAN performance. Compared in their benchmark are four CSPs: Amazon AWS, Microsoft Azure, Google AppEngine, and Rackspace CloudServers.

Hu, Zhu, Ardi et al. [68] use tens of PlanetLab–based VPs to periodically (every 10–11 minutes) issue ICMP and HTTP probes directly against Amazon and Microsoft VMs and storage resources. They then use the response error codes to discern failure cases and estimate Cloud availability. They define a probability model (Equation 3.1) to estimate the false outage rate with an increasing number of probes sent as retries when previous requests time–out. They also note that active probing can both overestimate and

underestimate Cloud availability – often depending on the treatment of this protocol by CSP DC policies.

$$Pr(\text{outage}|k \text{ probes}) = (p_{\text{loss}} + (1 - p_{\text{loss}}) \cdot p_{\text{loss}})^k \tag{3.1}$$

3.2 Cloud Network Profiling

Cloud Computing services, based on processing in remote DCs, exhibit latency and performance variations, which are a compound result of various components of the intermediate communication, computation and remote data storage. Measuring and evaluating performance of a Cloud service are thus highly ambiguous tasks. Nevertheless, methods of capture, analysis and interpretation of such variations, ought to be useful especially to Cloud tenants, who can thus discern normal and anomalous Cloud behaviors, and, as such, overturn the information deficit stemming from a very limited access to the CSP infrastructure.

Benson, Akella, Maltz [46] and Kandula, Sengupta, Greenberg et al. [72] both analyzed the nature of the intra–DC traffic, aiming to reveal traffic patterns and congestion conditions. Among interesting observations is the traffic arrival process at the edge switches that exhibits ON/OFF intermittent nature, where the ON/OFF durations exhibit heavy–tailed distributions. Also, in some DCs, a small but significant fraction of core links appears to be persistently congested, but there is enough spare capacity in the core to alleviate congestion. Also observed was that losses on the lightly–utilized links can be attributed to the bursty nature of the underlying applications inside the DCs.

Mizrahi and Moses [97] have studied end–to–end ICMP packet latency of Amazon AWS and Microsoft Azure CSPs in the time and frequency domains. They show that RTT measurements less then one second apart are autocorrelated. They also show that conventional long–tailed distributions do not fit the measured RTT distributions. And, using *Power Spectral Density (PSD)*, they show that RTT frequency components exist at high frequencies, which explains the temporarily high latency values recurring after tens–of–seconds timeframes.

Anomaly detection is a well–known concept in telecommunications and networking. We next present methods that use data from passive link measurements, RTT time-series and datasets combined from various intra–DC sources.

Lakhina, Crovella and Diot [79] have presented a *PCA subspace method* to detect, identify source of and quantify network–wide traffic anomalies. It works by separating the high–dimensional space of link traffic SNMP measurements into disjoint subspaces corresponding to normal and anomalous network conditions. The space decomposition of temporal link traffic measurement matrix Y into a set of m principal components $\{\vec{v}_i\}_{i=1}^{m}$ is shown in Equations 3.2 and 3.3.

$$\mathbf{v}_1 = \underset{\|\mathbf{v}\|=1}{\operatorname{argmax}} \|\mathbf{Y}\mathbf{v}\| \tag{3.2}$$

$$\mathbf{v}_k = \underset{\|\mathbf{v}\|=1}{\operatorname{argmax}} \|(\mathbf{Y} - \sum_{i=1}^{k-1} \mathbf{Y}\mathbf{v}_i\mathbf{v}_i^T)\mathbf{v}\| \tag{3.3}$$

Normal $\mathcal{S}$ and anomalous $\bar{\mathcal{S}}$ links are discriminated using magnitude of projection onto principal axes. Then, traffic on each link gets decomposed to normal $\vec{y}^n$ and anomalous $\vec{y}$ components, by projecting link traffic onto $\mathcal{S}$ and $\bar{\mathcal{S}}$ subspaces. The occurrence of a volume anomaly will tend to result in a large change to $\vec{y}$, determined using squared prediction error, tested against *Q–statistic*. The method is general enough to diagnose anomalies of various kinds, such as volume–based anomalies.

Zhang, Zhang, Pai et al. [133] focus on detecting and quantifying network path anomalies. They have built a PlanetLab–based monitoring system, *PlanetSeer*, which passively monitors traffic between PlanetLab and thousands of clients to detect anomalous behavior, using changes in flow TTL field and TCP timeout rate. PlanetSeer then coordinates active `traceroute`–based probes from many PlanetLab sites to confirm the anomaly, characterize it and determine its scope.

Krishnamurthy, Sen, Zhang et al. [75] propose building compact summaries of the traffic data using the notion of *sketches*, since keeping per–flow state is often considered expensive or slow. The linearity property of sketches enables to summarize traffic at various levels and implement a variety of timeseries–forecasting models (*ARIMA*, *Holt–Winters* and various *moving averages*) on top of such summaries. Significant changes are then detected by looking for flows with large forecast errors.

Forecast errors for general anomaly detection are also used by Brutlag [49]. He constructs a model of timeseries, forecasts a future datapoint and compares the observations to the predictions. If too many observations within a temporal window are deviant, an alarm goes off.

The *Holt–Winters forecasting* model is used by both Brutlag [49] and, in a slightly different non–seasonal form, by Krishnamurthy, Sen, Zhang et al. [75]. Holt–Winters forecasting uses a prediction (Equation 3.4) composed of baseline alias intercept, linear trend alias slope and seasonal trend (Equations 3.5, 3.6 and 3.7, respectively).

$$\hat{y}_{t+1} = a_t + b_t + c_{t+1-m} \tag{3.4}$$

$$a_t = \alpha(y_t - c_{t-m}) + (1-\alpha)(a_{t-1} + b_{t-1}) \tag{3.5}$$

$$b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1} \tag{3.6}$$

$$c_t = \gamma(y_t - a_t) + (1-\gamma)c_{t-m} \tag{3.7}$$

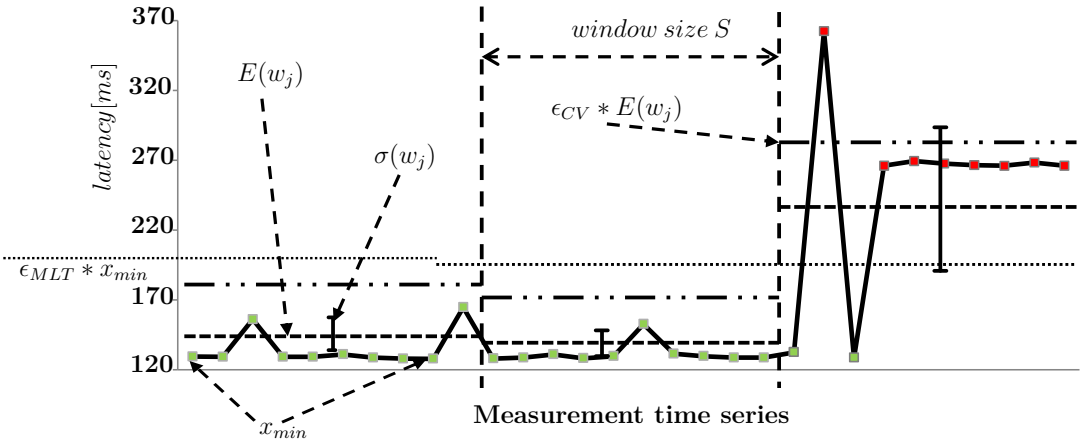
Mulinka and Kencl [99] have defined statistical metrics for profiling Cloud RTT timeseries in order to detect suspicious events (see Figure 3.3). The detection works as a sliding window, checking for significant changes in RTT–derived metric values (i.e., maximum value, coefficient of variation and histograms). They overcome the lack of the definition of normal Cloud behavior by using empirically–derived optimal parameter values to configure the model.

3.3 Cloud Service Benchmarking

With the ever–increasing trend of migration of applications to the Cloud and Fog environments, there is a growing need to thoroughly evaluate quality of the Cloud service itself, before deciding upon a hosting provider. Cloud–service benchmarking is difficult though, due to the complex nature of the Cloud Computing setups, the diversity of locations, variety of applications and their specific service quality requirements. However, such comparison may be crucial for decision–making and for troubleshooting of services offered by the intermediate tenants.

Folkerts, Alexandrov, Sachs et al. [63] describe the general properties an ideal benchmarking methodology should have. They also list use cases, where benchmarking is highly applicable.

The prevalent *cross–sectional studies* and benchmarking methodologies provide only a shallow comparison of Cloud services, whereas state–of–the–art tooling for specific comparisons of application–performance parameters, such as latency, is insufficient.



a)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
MAXIMUM LATENCY THRESHOLD METRIC																														
$x'_i[boolean]$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
$y_j[boolean]$	0										0										1									

b)

CV METRIC			
$\sigma(w_j)[ms]$	12.7	7.1	65.8
$E(w_j)[ms]$	135.4	131.7	249.4
cv	0.094	0.054	0.264
$y_j[boolean]$	0	0	1

c)

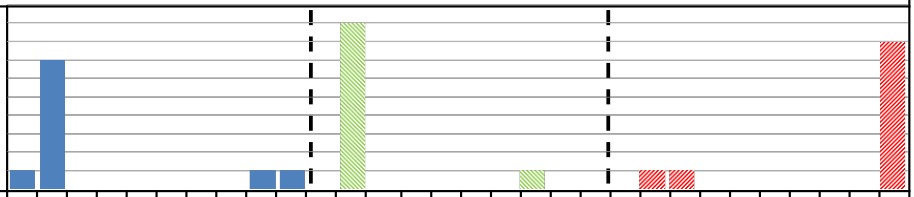
HISTOGRAM METRIC																														
H_{TEST}																														
$B = 10$	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
p_{LOW}, p_{HIGH}	128, 157										128, 157										128.3, 151.8									
H_{TRAIN}	1	7	0	0	0	0	0	0	1	1	1	7	0	0	0	0	0	0	1	1	1	6	0	1	0	0	0	0	1	1
H_{TEST}	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	8
$y_j[boolean]$	N/A										0										1									

Figure 3.3: ***RTT timeseries profiling for suspicious event detection [99].*** *Three metrics (a), (b) and (c) independently flag a window as anomalous or not and these partial verdicts y can then be reconciled into an ultimate decision regarding window suspicion.*

Cross–sectional studies (also called *breadth and snapshot*) allow quick shallow benchmarking of Cloud resources via a test suite. Li, Yang, Kandula et al. [86] have presented *CloudCmp* comparator that measures computation, storage and network resources, the last using a TCP throughput and end–to–end ICMP latency.

Chhetri, Chichin, Vo et al. have presented *Smart CloudBench* framework [53] that deploys a standard transactional web benchmarking suite TPC–W and brings about representative load conditions. They then measure latencies and record error codes in order to estimate the cost–to–performance ratio. Based on computed scores, CSPs get ranked by *utility theory* and *preference policies*.

Lenk, Menzel, Lipsky et al. [84] have created a reference Cloud performance measurement VMs for various Cloud usage scenarios. They benchmark IaaS offerings by considering type of service running on a VM. They show that performance indicators from CSPs are not sufficient and VM performance can vary every time it is restarted.

Bocchi, Mellia, Sarni [47], as part of their Cloud storage benchmarking suite, report RTTs to DCs of major Cloud storage providers. Their data confirm artificial limitations imposed by CSPs, throughput variations across DCs and, in some cases, performance domination of inter–continental data transfers over regional ones.

Menzel and Ranjan [95] have presented *CloudGenius* tool, which independently assesses a suitability of the offered VM images A and infrastructure services S using requirements R and numerical attributes $\hat{A}$ and non–numerical attributes $\hat{B}$ (Equations 3.8 and 3.9). Feasible combinations are created using Equation 3.10. Then, using user–specified preferences w_a and w_s and image–service dependencies D, a composite decision is made that yields the optimal solution (Equations 3.11 and 3.12).

$$f(a_i, \hat{A}_{a_i}, \hat{B}_{a_i}) = \begin{cases} \sum_{j=0}^{|\hat{A}_{a_i}|} w_j \chi(\alpha_{j,a_i}) & \forall r \in R_A : r = \text{true} \\ 0 & \text{else} \end{cases} \mapsto v_{a_i} \tag{3.8}$$

$$g(s_j, \hat{A}_{s_j}, \hat{B}_{s_j}) = \begin{cases} \sum_{i=0}^{|\hat{A}_{s_j}|} w_i \chi(\alpha_{i,s_j}) & \forall r \in R_S : r = \text{true} \\ 0 & \text{else} \end{cases} \mapsto v_{s_j} \tag{3.9}$$

$$v_{a_i,s_j} = b : (a_i, s_j) \mapsto f(a_i, \hat{A}_{a_i}, \hat{B}_{a_i}) \bullet g(s_j, \hat{A}_{s_j}, \hat{B}_{s_j}) \quad \forall i, j : (a_i, s_j) \in D \tag{3.10}$$

$$b(a_i, s_j) = \begin{cases} w_a * f(\cdot) + w_s * g(\cdot) & (a_i, s_j) \in D \\ 0 & \text{else} \end{cases} \mapsto v_{a_i,s_j} \tag{3.11}$$

$$\max \ \ b(a_i, s_j) = \max \ \{v_{a_1,s_1}, \ldots, v_{a_m,s_n}\} \tag{3.12}$$

Limitations surrounding the general and presented cross–sectional studies include: shallow nature that provides a little or no explanation, restrictions by CSP often lead to invalid comparisons and a short timeframe that leads to inaccuracies. Also, they often cannot answer questions related to the global complex distributed applications.

Collective Intelligence Benchmarking (CIB) represents a community–knowledge approach, which uses many datasets from distributed applications and 3$^{\text{rd}}$–party services in order to ascertain a performance baseline for a given service or application. Commercial vendors include *Indeni* [20] or *logz.io* [23].

3.4 Cloud Connectivity Optimization

Nowadays, people leverage, utilize and in some cases depend upon smart devices to solve problems [85]. Many such use cases are enabled by or benefit from a central service backend, increasingly hosted remotely in public Cloud DCs. This arrangement, however, poses a non–trivial task of ensuring end–to–end Cloud connection properties, necessary for devices to operate as expected. The reality is that even the most popular Cloud services often degrade and occasionally fail for various reasons [88]. This uncertainty regarding Cloud–service performance and reliability could be addressed by intelligent approaches, which continuously re–evaluate DC performance and assign Cloud requests to momentarily optimal DCs.

3.4.1 ISP and CSP Perspective

Sun, Yu, Anand et al. [115] abstract inter–related Cloud applications using a virtual network and strive to accommodate it using optimal provisioning under the current resource state of DC. They use link revenue (difference between sum of income from allocated bandwidth resources x and sum of bandwidth costs of traffic flows fl) and server revenue (difference between sum of server resource incomes p_n and sum of server resource costs cn) metrics inside Objective function 3.13. Furthermore, they present a genetic algorithm heuristic for addressing a situation with unsplittable flows.

$$\max \left(\sum_{p \in P, s=\text{src}(p), t=\text{dst}(p)} \sum_{i} x_{st}^{i} p_b - \sum_{e \in E_s} \text{fl}_e \text{cl}_e \right) + \left(\sum_{m \in V_F} \text{rn}_m p_n - \sum_{m \in V_F} \text{rn}_m \text{cn}_m \right) \quad (3.13)$$

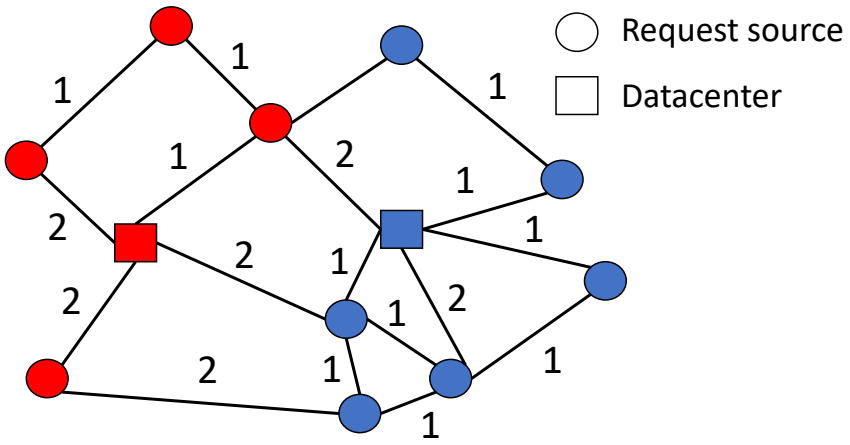


Figure 3.4: ***Voronoi tessellation of weighted graph of DCs and request sources [61].***

Doyle, Shorten and O'Mahony [61] have presented *Stratus* system that employs *Voronoi tessellation* to assign Cloud requests to optimal DCs. Graph edges are weighted using a composite metric of three weighted objectives: carbon emissions G, latency T and electricity cost E (Equation 3.14). Voronoi partitions are then used to minimize the distance between the sources of requests and the DCs (Objective function 3.15). An example situation is shown in Figure 3.4.

$$w_i = T_i + R_1(G_i) + R_2(E_i) \tag{3.14}$$

$$\min \sum_{i=1}^{N} \sum_{j \in P} d(i,j) \tag{3.15}$$

Couto, Secci, Campista et al. [56] design geo–distributed DCs using disparate, but important metrics – latency l and survivability s (i.e., a capacity of a system to operate after failures). They formulate the problem as MILP with Objective function 3.16 and show significant latency increase only in the case of big survivability requirements.

$$\max(1-\beta)s - \beta\frac{l}{L_{\max}} \tag{3.16}$$

Forestiero, Mastroianni, Meo et al. [64] have presented a hierarchical approach for inter–Cloud infrastructures, which preserves the autonomy of respective DCs and at the same time allows for an integrated management of heterogeneous platforms. They use carbon emissions, energy utilization and electricity cost as metrics for their multi–site assignment algorithm.

Table 3.1: ***Cloud connectivity optimization metrics from literature.***

Paper	Composite metric
Sun et al. [115]	server revenue, link revenue
Doyle et al. [61]	carbon emission, latency, electricity cost
Couto et al. [56]	latency, survivability
Forestiero et al. [64]	carbon emission, energy utilization, electricity cost
Tripathi et al. [122]	server cost, renewables cost, brown energy cost
Zhou et al. [134]	latency, computation cost
Maswood et al. [92]	bandwidth cost, link utilization, resource utilization
Tomanek et al. [119]	mean latency, latency deviation, timeout rate

Tripathi, Vignesh and Tamarapalli [122] use server cost, renewables cost and brown energy cost in their capacity planning algorithm. They show that capacity provisioning that considers green energy integration, not only lowers carbon footprint, but also reduces the Total Cost of Ownership (TCO).

Zhou, He, Cheng et al. [134] provision resources for scientific workloads using latency and computation costs. Their engine for generation of resource provisioning plan uses probabilistic optimization and combines high–level declarative workflow optimization goals with the user–specified latency and monetary cost constraints.

Maswood and Medhi [92] assume ISP–operated DCs d, for which a request assignment at ISP PoP i improves by considering bandwidth cost u, link utilization l and resource utilization k (reflected by weighted terms in Objective function 3.17).

$$\min \alpha \sum_{i \in I} \sum_{d \in D} \sum_{l \in L} \tilde{z}_{il}^{d} + \mu u + \gamma \sum_{d \in D} k_d \tag{3.17}$$

The said optimization techniques operate at CSP level and are usually formulated as Mixed–Integer Linear Program (MILP). They score DCs using composite metrics, reflecting objectives geared towards greening, user experience or cost (Table 3.1). Performance evaluation is often carried out using hypothetical scenarios and varying weights of partial objectives rather than reporting and validating improvement by instant deployment, using the actual Cloud behavior ground truth. Also, effectiveness of the techniques strongly depends on composite metric's weight settings, which is tricky to get right in a real environment and, unfortunately, only a little guidance is provided.

Instead of the full integer programming, Toosi, Qu, de Assunção et al. [121] employ a simpler change–point detection approach that adapts load distribution among DCs to dynamic factors like electricity prices or availability of renewables (Algorithm 1).

Algorithm 1 Green load–balancing policy [121]

```
R ← 0
for all DCs d in the list do
    c ← DC's energy consumption in Watt–hour within the time window
    t ← number of requests redirected to the site within the same time window
    a ← currently available renewable power at the site in Watt
    w ← Watt–hour consumption per request (c/t)
    r_d ← request rate (# reqs/h) DC d can accommodate using renewables (a/w)
    R ← R + r_d
end for
γ ← request rate (# reqs/h) at Global–LB
if γ < R then
    for all DCs d in the list do
        set weight as r_d/R
    end for
else
    find the DC d' with the cheapest price of brown energy per request
    L ← γ
    for all DCs d in the list except d' do
        set weight as r_d/γ
        L ← L − r
    end for
    set the weight for d' as L/γ
end if
update HAProxy weights accordingly
```

A fair comparison of all these disparate state–of–the–art solutions is much needed in order to reveal importance of various objectives in various contexts and, consequently, maximize benefits at an acceptable complexity. What makes the comparison difficult is a mixture of vendor lock–in and request–level nature, incompatibility of objectives, unavailability of ground truth and operation at a CSP level.

3.4.2 Edge Perspective

At the network edge, smart technology has been increasingly deployed [118], leading to a growing network traffic and diverse applications (e.g., control, telemetry, entertainment or update). Many popular smart appliances have strict connection–quality requirements to remain operational – *Nest* thermostat's availability [26] or *Echo* speech–trigger's latency [16] requirements to name a few. Some smart–service providers employ an uplink monitoring from appliance–end or service–end, but others do not implement any such provisions and rely on well–behaved or application–aware networks, like in the case of home and mobile edge networks [69].

Karagiannis, Gkantsidis, Key et al. [73] have proposed a cooperative host–based system *HomeMaestro* to manage home and small networks by monitoring local and global LAN application performance and detecting contention for network resources by correlating performance metrics across flows and hosts.

IoT data from the appliances are often handled by data streaming and processing frameworks like *Apache Kafka* [21], *Google Millwheel* [38] or *Amazon Kinesis* [22], which buffer and transform telemetry data inside LAN or at the edge before sending the stream to its consumers. These consumers reside either in Cloud DCs or at the edge [48], like in the case of major–provider offerings *Amazon Greengrass* [17] or *Azure IoT Edge* [5]. These complex service architectures and computation–offload needs contributed to the evolution of the conventional home routers into smart gateways [109] that are resource rich and have extra capabilities.

3.5 Conclusion

In this Chapter, we have reviewed the previous research and development in the areas relevant to this thesis, namely Cloud–service network latency monitoring, Cloud network profiling, Cloud–service benchmarking and Cloud–service connectivity optimization.

The gist of the existing Cloud latency monitoring techniques is largely similar. The tricky part seems to be the best possible reconciliation of methodology transparency, cost of deployment, measurement frequency, global reach and data dependability. These ought to be achieved via rigorous methodology and measurement architecture.

Many anomalous Cloud behaviors and valuable insights still go by unnoticed, suggesting that conventional network profiling and traffic characterization still have a room to provide an increased coverage. Cloud benchmarking state–of–the–art clearly shows a lack of longitudinal (a.k.a. *deep and continuous*) techniques that consider underlying–service performance variations and accommodate niche applications. Furthermore, Cloud service issues and performance variations at various levels and scales of Cloud infrastructure confirm the need for Cloud connectivity optimization.

Some conclusions with respect to the topic of this thesis can be reached based on the works reviewed. It is the goal of this thesis to fill the aforementioned gaps by developing methodologies that preserve the advantages and overcome the drawbacks of the state–of–the–art techniques.

Chapter 4

Monitoring Methodology

4.1 Introduction

In this Chapter, we describe the terminology of our distributed *multidimensional Cloud–latency monitoring methodology*. We then show why and how much it increases the accuracy of Cloud latency measurements.

4.2 Terminology

By *measurements* we understand RTTs of protocol message exchanges, processing times and the entire end–to–end latencies. By *multidimensional* we understand measurements capable of being looked at from perspectives of multiple dimensions, i.e., VPs, DCs, communication protocol layers etc. Using a terminology that follows, a monitoring dataset can be viewed as a set of timeseries described as:

$$X^{L,V,Q,F,B,P} = \{x_n^{l,v,q,f,b,p}\},\ n \in [1,2,\ldots,N]\,, k \in [1,2,\ldots,K]\,, n \equiv 0 \bmod k \qquad (4.1)$$

A timeseries $\vec{x}$ contains N single multidimensional RTT measurements x_n, where every successive k measurements belong to a probe train captured around the common time instant. A measurement x_n has the following dimensions:

1. Measured protocol layer of the network communication stack $l \in L$ (e.g., TCP);
2. Internet Vantage Point location $v \in V$ (e.g., end host in Prague);
3. Internet Vantage Point designation $q \in Q$ (e.g., secondary);

4. Measured frontend–resource DC location $f \in F$ (e.g., Web server in Dublin);

5. Measured backend–resource DC location $b \in B$ (e.g., SQL server in Singapore);

6. Cloud Service Provider $p \in P$ (e.g., Microsoft Azure).

The entire idea, with dimensions instantiated, is captured in Figure 4.1. Note that two situations may arise. First, in the case of frontend resource measurement, b takes no value. Second, in the case of backend resource measurement, v takes no value. This is necessary, because we assume that backend resources are generally not directly accessible from the Internet.

4.3 Performance Evaluation

In this Section, we evaluate the methodology of multidimensional latency monitoring by using the accuracy of Cloud–service availability estimates, since outage reports are usually the only behavioral ground–truth data CSPs disclose publicly [15]. For that purpose, we adapt the analyses by Naldi [100] and by Hu, Zhu, Ardi et al. [68]. The former author has evaluated how the redundant ICMP probes within a measurement probe train decrease the number of detected false–positive Cloud outages. The latter authors have evaluated the precision with which retried measurements and additional control–measurement sites decrease the number of detected false–positive Cloud outages and also

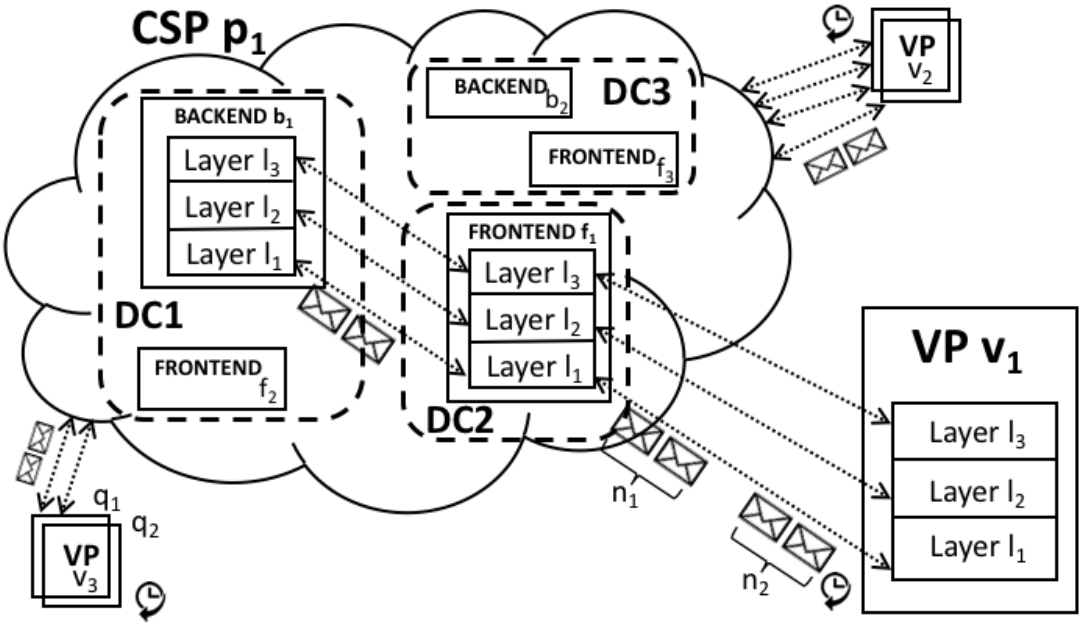


Figure 4.1: *Multidimensional monitoring arrangement.*

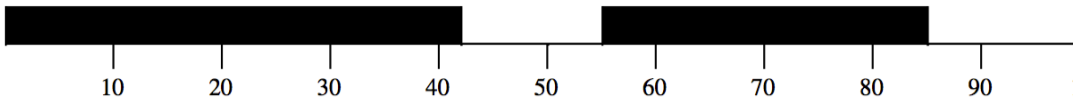


Figure 4.2: ***State transitions between outage and normal periods*** *[100].*

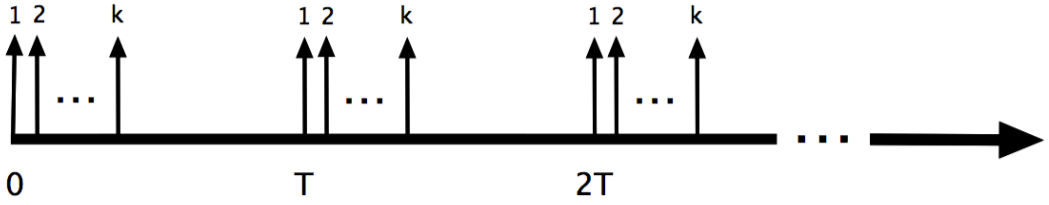


Figure 4.3: ***Probe trains*** *[100].*

how accuracy improves with measurements over upper–layer communication protocols. Our data capture platform (despite published earlier) is a generalization of their ideas and has been used in applications beyond Cloud availability estimation. We paraphrase some and conduct other of their suggested analyses, to derive suitable sizes of monitoring dimensions to be used in the rest of this thesis.

4.3.1 ICMP Probe Trains

Naldi [100] notes that the process of availability measurement using active redundant ICMP probing has impact on the obtained numeric result and emphasizes different possible failures, including those the user is responsible to recover from (near–end routing, DNS lookup, packet loss, rate limiting). Availability A of a system, which undergoes phases of normal operation separated by outages (Figure 4.2) is defined as a ratio between the uptime and the total time. Using RTT measurements, it can be approximated by a ratio of timed–out measurements to all measurements:

$$A = \frac{\text{Uptime}}{\text{Total time}} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}} = 1 - \frac{\text{Number of timeouts}}{\text{Total measurements}} \tag{4.2}$$

ICMP often does not measure the end–to–end service per se, but likely a service frontend or a DC edge responder (see Figure 2.8). Also, ICMP messages can be filtered and, hence, never answered. Therefore, upper–layer protocols that test the end–to–end system and are not depreferentially treated by the DC should be used together with ICMP.

ICMP messages should not be sent in isolation, but instead as probe trains to achieve a better accuracy in the face of temporary glitches. As shown in Figure 4.3, the tester sends off a probe train of $1 \leq k \leq K$ probes in a short timeframe, repeating every T seconds. To compare the observed availability against the SLA commitments, an observation window of length C is considered. At the end of each probing round of k measurements, the tester outputs an availability statement concerning the Cloud, declaring the Cloud service either available or not. That statement is valid till the next probing round, when a new availability statement comes out. The observation window can therefore be considered as $B = \lfloor C/T \rfloor$ time blocks, such that a number of availability statements falls into the $[0, B]$ range, where the lowest value 0 represents a service deemed 100% unavailable over the observation window, while the largest possible value of B represents a 100% availability. If N_{out} denotes the number of time blocks deemed unavailable, the availability estimate can be derived using Equation 4.3.

$$\hat{A} = 1 - \frac{N_{\text{out}}}{B} = 1 - \frac{N_{\text{out}}}{\lfloor C/T \rfloor} \tag{4.3}$$

In order to correctly estimate large availability figures, the number of blocks must be correspondingly large, otherwise the granularity due to the blocks will mask short unavailability periods. For example, if $B = 100$, the next largest availability figure to 100% that can be estimated is $99/100 = 0.99$, i.e., a 2–nine availability. If availability figures as large as four nines are to be estimated ($A = 0.9999$), the minimum number of blocks must be 10000. In order to achieve that number, given $T = 10$ minutes, the observation window must be at least $10 \cdot 10000 = 100000$ minutes long, which corresponds to slightly more than 69 days. In general, to measure availability A, the observation window size C condition in Inequality 4.4 must hold.

$$C \geq \frac{T}{1 - A} \tag{4.4}$$

In order to derive an availability statement for any single train of k probes, Naldi [100] considers Majority Voting criterion, which declares an outage if a majority of probes receives no responses.

Single Probe

Naldi [100] first analyzes behavior of the Cloud using a single probe. The actual outcome of the measurement process is impacted by both the network and the Cloud failures, so an outage may be declared even when the Cloud is available (false outage).

If r is the true Cloud outage probability and p is the packet loss probability on the Cloud path (Naldi assumes identical probability in both directions and failures on the two trips uncorrelated due to time spacing), the probability of missing a response to a single probe P_{mr} is given by Equation 4.5 and shown in Figure 4.4.

$$P_{\text{mr}} = p + (1-p)r + (1-p)(1-r)p \simeq 2p + r \tag{4.5}$$

The biasing factor P_{mr}/r may be large, as shown in Figure 4.5, especially when $p > r$. This general case may be simplified by looking at the two cases where the Cloud is either available or not. Focusing on the former case, negative outcome rises and just one case (both trips occurring with no packet loss) is reported as successful. Since the Cloud is available, what is detected as a failure is actually a false outage. By marking the status of the Cloud by a flag variable X ($X = 1$ if the Cloud is available and 0 otherwise), and the outcome of the probe test by another flag variable Y ($Y = 1$ if receiving a response and 0 otherwise), the probability of a missing response, conditional to the Cloud availability

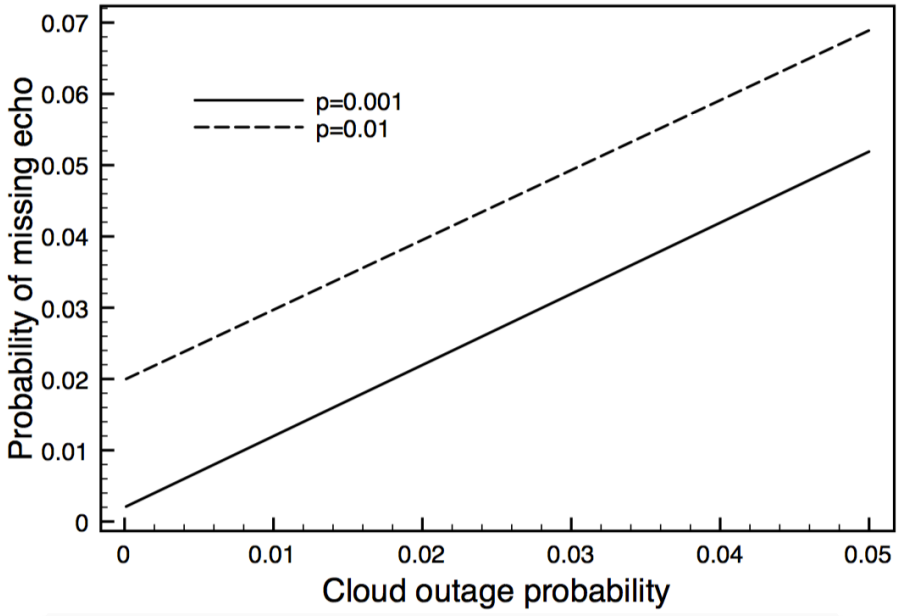


Figure 4.4: ***Probability of missing a response*** *[100].*

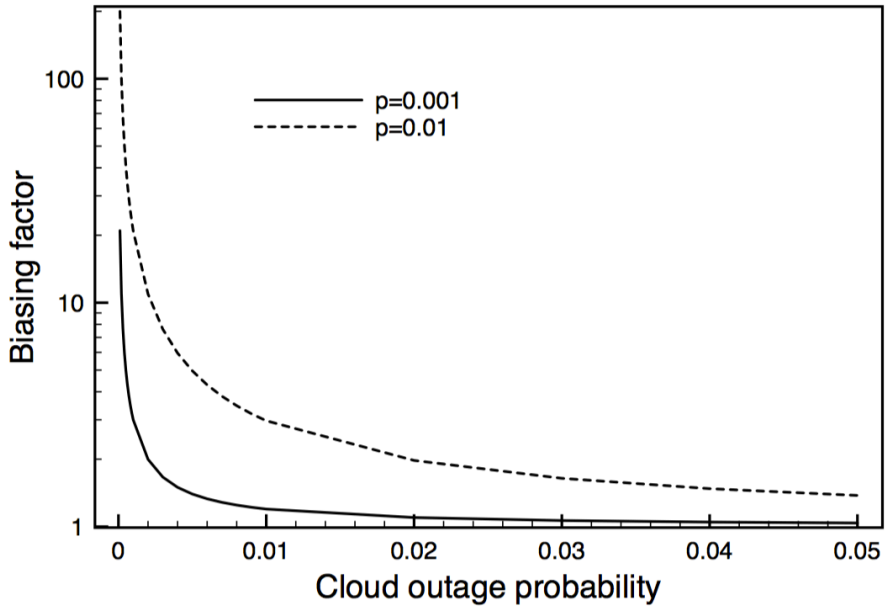


Figure 4.5: ***Biasing factor*** *[100].*

using a single probe, is given by Equation 4.6.

$$P_{\text{mrca}} = P[Y = 0|X = 1] = p + (1 - p)p \simeq 2p \tag{4.6}$$

Probe Train

Using a train of $1 \leq k \leq K$ probes, Naldi [100] then evaluates the probability of declaring a false outage when the Cloud is perfectly available.

The criterion to output an availability statement is *Majority voting*, as it was shown to be better suited than *Unanimous positive* and *Unanimous negative voting* [100]. According to the Majority voting, a Cloud outage is declared after a train of k probes when there are at least $k_{min} = \left\lceil \frac{k+1}{2} \right\rceil$ missed responses. Naldi assumes that the outcomes of successive probes in the face of network failures are uncorrelated. The number of missed responses in a probe train of k requests therefore follows *Binomial distribution* with parameters p and k. False outage probability P_{fo} given k probes is then denoted by Equation 4.7.

$$P_{\text{fo}}(k) = \sum_{i=k_{\min}}^{k} \binom{i}{k} P_{\text{mrca}}^{i} (1 - P_{\text{mrca}})^{k-i} \tag{4.7}$$

Figure 4.6 shows that, for $k = 9$, the repetition mechanism is highly effective, i.e., probability of false outage is 8.10^{-4} even under high packet loss probability ($p = 0.05$).

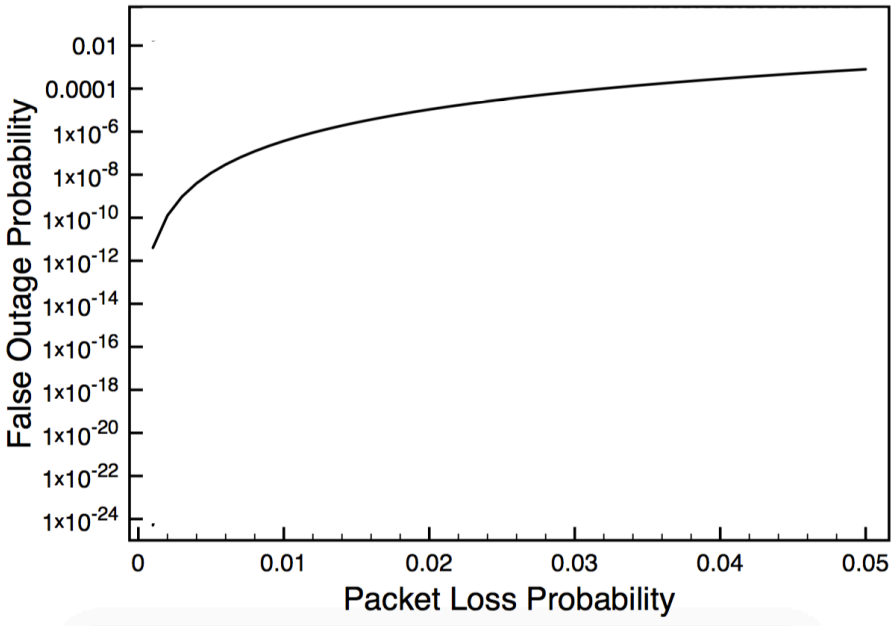


Figure 4.6: ***Probability of false outage*** $(k = 9)$ *[100].*

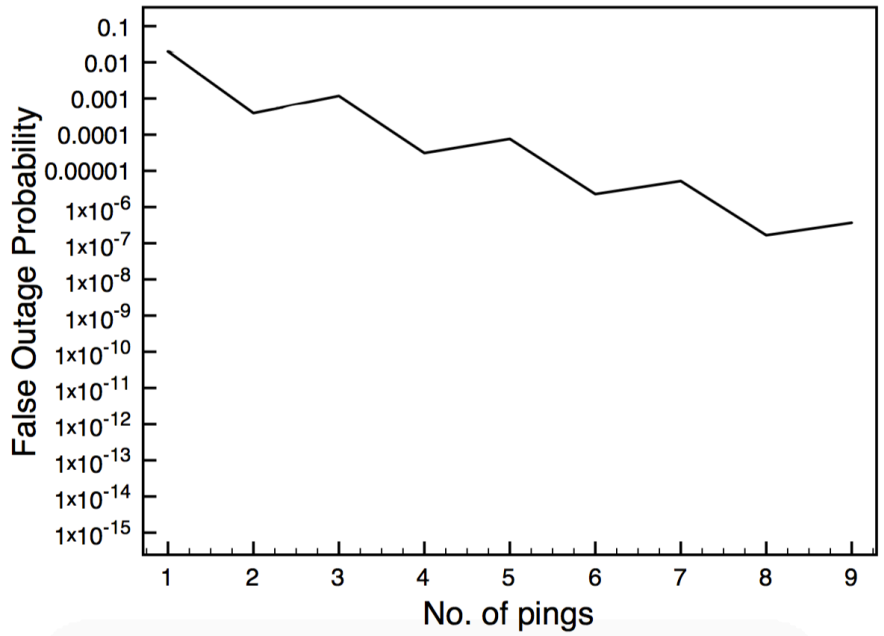


Figure 4.7: ***Impact of retries*** $(p = 0.01)$ *[100].*

Size of the probe train has a significant impact. In Figure 4.7, plotted for the packet loss probability $p = 0.01$, is a soft descent of false outage probability, whose staircase–like appearance is an artifact due to the Majority rule (e.g., when passing from 4 to 5 probes the useful cases are 3 or 4 out of 4; but 3, 4, or 5 out of 5, respectively).

The figures are valid as long as the probing–round period does not become similar to the outage duration. By marking the occurrence of the measurement instant preceding the outage as time 0, and the next probing round taking place at time T, the outage will take place at a random time O, $0 \leq O \leq T$. If O is considered to be uniformly distributed, the failure will not be detected if the recovery from the outage is achieved before the next probing round. If the outage duration is L, that condition can be expressed as $O + L < T$. The probability that the outage goes undetected is then denoted using Equation 4.8.

$$P_{\text{undet}} = \mathbb{P}[O + L < T] = \mathbb{P}\left[\frac{O}{T} < 1 - \frac{L}{T}\right] = \begin{cases} 0 & \text{if } T \leq L \\ 1 - \frac{L}{T} & \text{if } T > L \end{cases} \tag{4.8}$$

This resulting probability of no detection, given by Equation 4.8, is plotted in Figure 4.8. If outages go undetected, it could be difficult to measure SLA compliance. The number of failures is heavily distorted since short–lived outages may go undetected, and the number of long outages may be underestimated as well, unless the measurement frequency is sufficiently high.

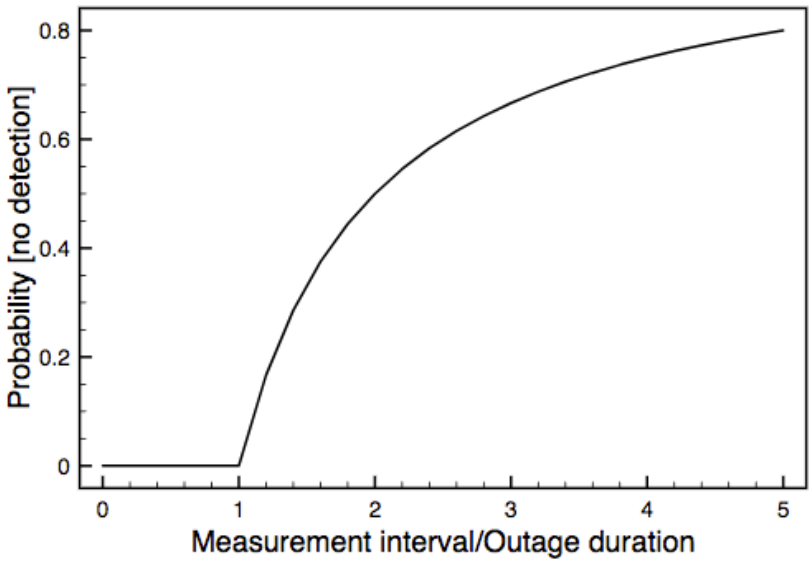


Figure 4.8: ***Probability of outage going undetected*** *[100].*

4.3.2 Multiple Protocol Layers

This Subsection evaluates the benefit of probing various protocol layers and demonstrates the benefit of additional control–measurement sites. Hu, Zhu, Ardi et al. [68] employ a *probe–retry* mechanism rather than *probe trains* in order to estimate Cloud availability. They show that both the lower and the upper–layer protocols can overestimate or underestimate Cloud VM and storage availability. Probing multiple layers is thus desirable for root cause investigation of Cloud failures and upper layer protocols should be preferred whenever probing multiple layers is not possible.

They first note that a probe rate of a few packets per second is low enough to avoid most rate limiters. Prior to the similar approach by Naldi [100], they evaluate the probability of falsely inferring an outage caused by random packet loss, as a function of packet loss rate (Figure 4.9). Here they assume k tries and declare the service down when all tries fail (resembling *Unanimous negative voting*). Using packet loss, they model the loss of the request or response using a simple analytic model in Equation 4.9.

$$Pr(\text{outage}|k \text{ probes}) = (p_{\text{loss}} + (1 - p_{\text{loss}}) \cdot p_{\text{loss}})^k \tag{4.9}$$

Without retries ($k = 1$), the false outage rate approximates the loss rate. As the wide area packet loss can be around 1%, measurement without retries will show false outages and skew estimates of Cloud availability. Fortunately, if packet loss is assumed to be independent, then a few retries drive the false outage rate well below typical Cloud

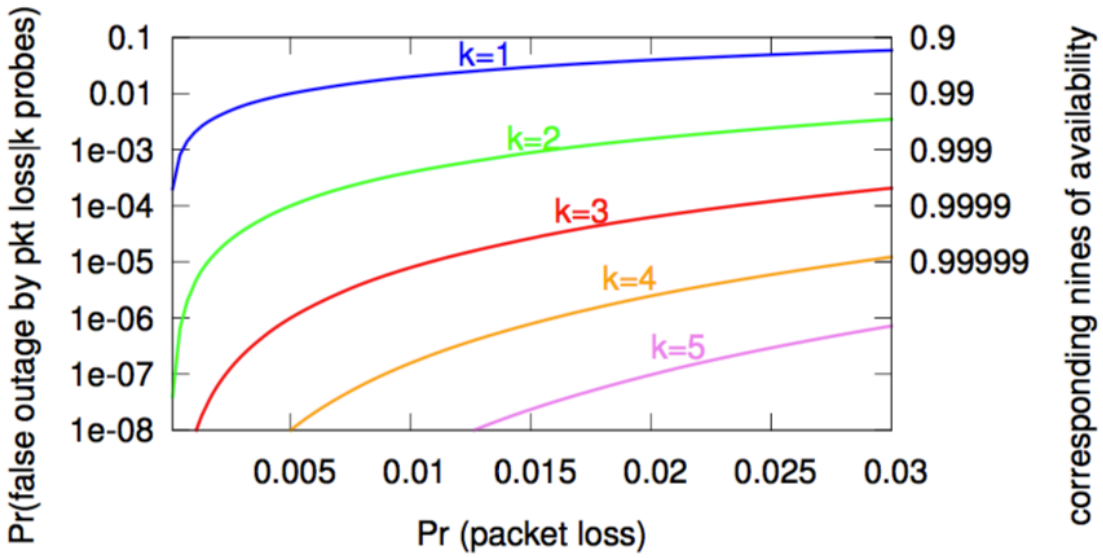


Figure 4.9: ***Probability of false positive caused by random packet loss*** *[68].*

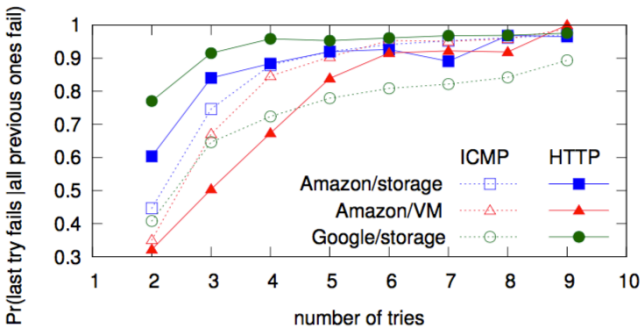
outage rates. For example, with three tries and 1% packet loss, message loss will be around 10^{-5}, or five nines of availability. If network loss rates are assumed to peak at a few percent, 4–6 tries are the appropriate sweet spot.

Hu, Zhu, Ardi et al. [68] then compare the model against experimental results for ICMP and HTTP (both for frontend and backend). The dotted lines in Figure 4.10 show the probability that the k–th try fails if all previous $k-1$ tries failed. They evaluate this by considering the first k tries from each observation. In the case of ICMP, retries clearly help. An initial loss is followed by a second loss only 35–45% of the time, so 55–65% of the time the second try succeeds, suggesting that the first try was random loss. This effect diminishes with more retries, generally plateauing around 5 or 6 probes.

In the case of HTTP, they retry at the application layer, but the kernel also does retries for the TCP connection (the HTTP client has a ten–second application timeout set). The OS does three SYN transmissions in this time, providing two network–layer retries for free for each application try. This benefit can be observed in the Figure 4.10a, where single–try HTTP loss rates are much lower than ICMP. Kernel–layer retries help even with application retries, as seen in Figure 4.10b, where the basic HTTP failure rate for Amazon storage and Google storage is half that of the ICMP. However, even HTTP benefits from multiple application–layer retries before the conditional benefit of additional tries plateaus. Application–layer probes show even higher levels of conditional failure than network–layer, with 50% of second HTTP attempts failing on average, presumably because of the additional kernel–layer retries. However, this result means that 50% of second attempts succeed, i.e., application–layer failures are sometimes transient and 5–6 probes are recommended also for the upper layer protocols.

	Pr(first try fails)	
Target	**ICMP**	**HTTP**
Amazon/VM	.00585	.00232
Amazon/storage	.00574	.00435
Google/storage	.00631	.00217

(a) *Table of probability that the first try fails*



(b) *Conditional probability that k–th retry fails, given failure of prior tries*

Figure 4.10: ***Comparison of loss and retries.*** *Comparing loss and reties for each target and protocol. Nine retries were used to rule out random loss [68].*

HTTP/ICMP comparison

Hu, Zhu, Ardi et al. [68] directly compare network and application–layer probes from tens of PlanetLab VPs to judge Cloud availability. They employ multiple control–measurement sites to rule out problems near primary VPs. This, together with sufficient retries, avoids effects of random packet loss and transient network issues in the middle, leaving outages at or near the CSP as the primary problem.

Cloud services are made up of Internet–facing frontends with sophisticated back-end clusters. As mentioned in Section 2.2, in some cases, ICMP may be handled by the frontends, while HTTP's end–to–end tests reach the backend. Hu, Zhu, Ardi et al. [68] then evaluate this difference. While the protocols almost always agree, there are many small disagreements (Figure 4.11). They next show several causes of disagreement using representative examples in strip charts, where each data column shows one probing round (with 24–hour boundaries as vertical black lines), and each pair of rows shows ICMP and HTTP observations from one VP (the upper, in blue, is ICMP and the lower, in red, is HTTP). Light colors represent successful probes, medium colors represent failures of some tries, but eventual success. Dark blue diamonds show ICMP outages (all ICMP tries fail) and dark red squares show HTTP outages (all HTTP tries fail). White areas show cases where one of the control–measurement sites failed to respond to either ICMP or HTTP or where data upload to the collection site fails.

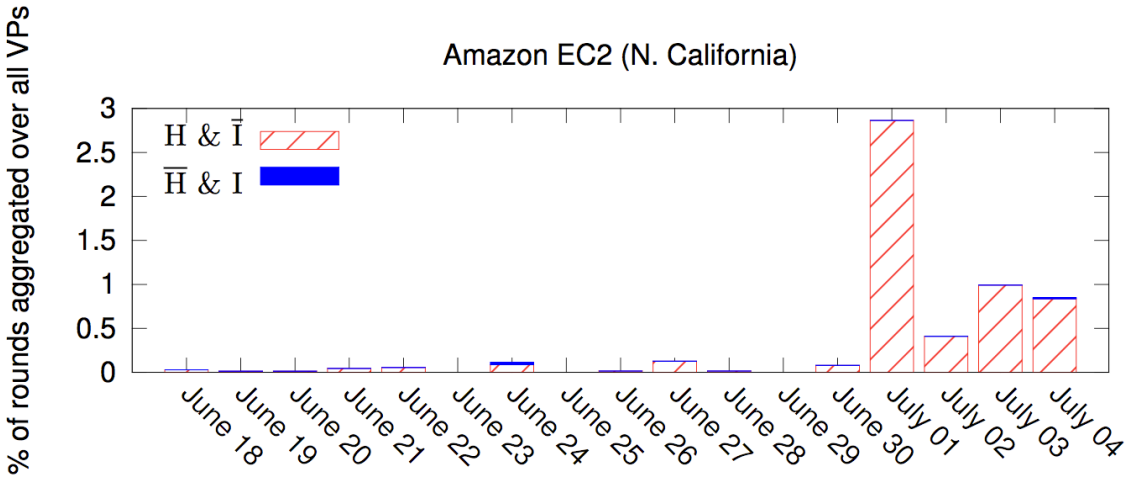


Figure 4.11: ***Quantifying disagreements between HTTP and ICMP probes.*** *This includes both the HTTP success and ICMP failure (red striped bar) and the HTTP failure and ICMP success (blue bar) [68].*

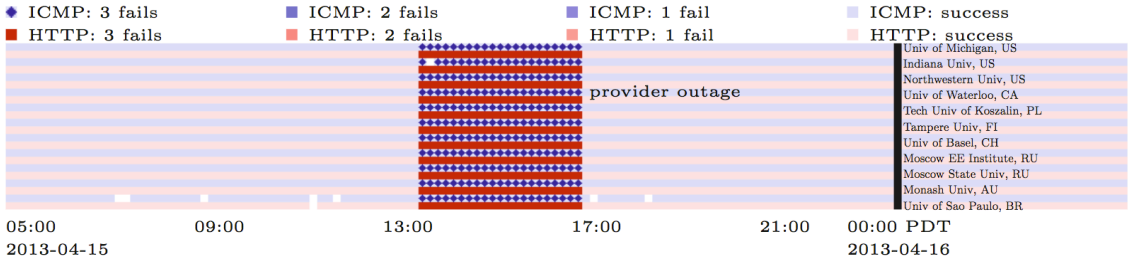


Figure 4.12: ***CSP–confirmed outage at Amazon EC2 Singapore site*** *[68].*

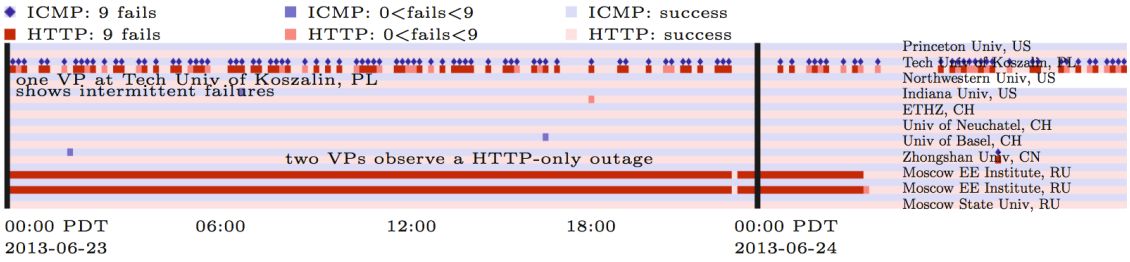


Figure 4.13: ***Intermittent failures from one VP to Amazon S3 in N. California*** *[68].*

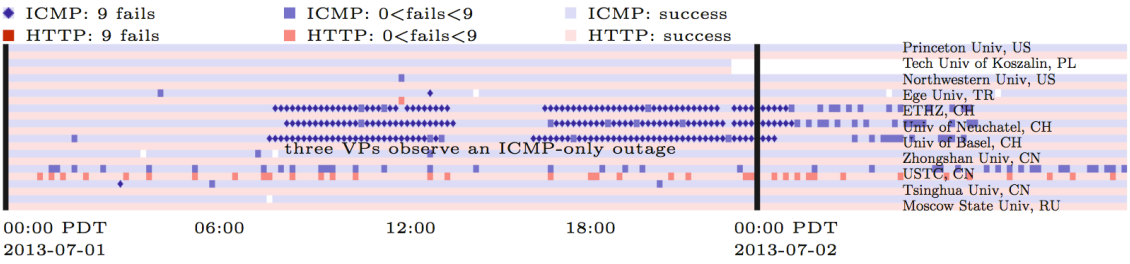


Figure 4.14: ***ICMP–only outage to Amazon VM in N. California*** *[68].*

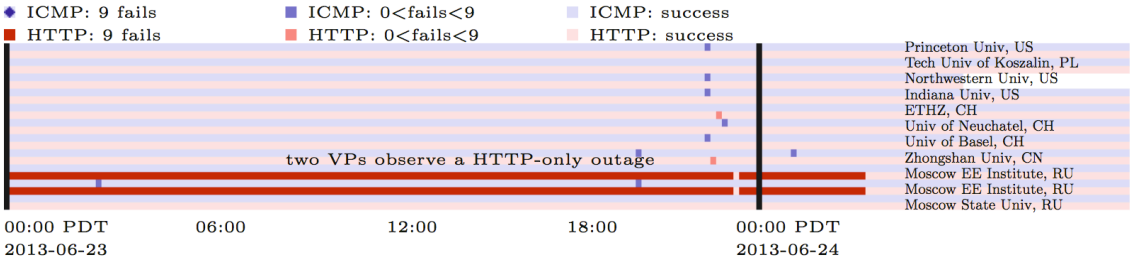


Figure 4.15: ***HTTP–only outage to Amazon VM in Singapore*** *[68].*

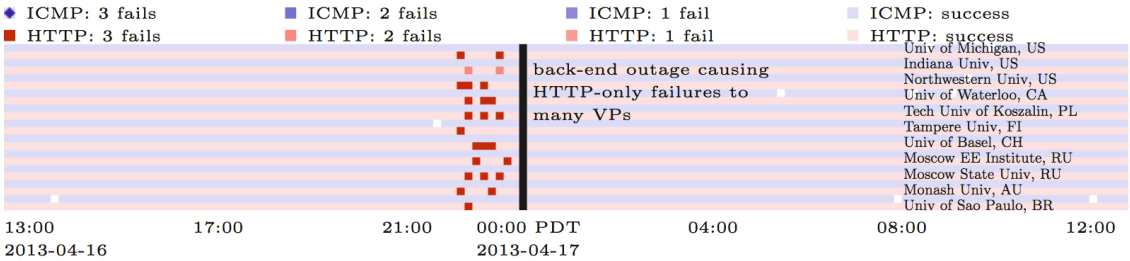


Figure 4.16: ***CSP–confirmed backend outage at Amazon S3 in Japan*** *[68].*

Figure 4.12 shows CSP–confirmed outage at one Amazon EC2 site, reported consistently by ICMP and HTTP. Figure 4.13 shows persistent problems between Polish VP and Amazon S3 site, as both ICMP and HTTP report outages. Figure 4.14 shows a likely route change to a route with reverse–path ICMP filtering to Switzerland. Figures 4.13 and 4.15 show HTTP–only outage to either Amazon S3 or EC2. Figure 4.16 shows a likely Amazon S3 backend failure, because ICMP reports storage frontend operating normally.

4.3.3 Real World Verification over Long Term

We have analyzed the behavior of a periodic probe train at TCP and HTTP layers over a long term using real CSP service measurements. Specifically, we have conducted the evaluation of false outage probability, using analyses by Naldi [100] and Hu, Zhu, Ardi et al. [68], suggested earlier in this Chapter. We use a 70–day RTT dataset of two major CSPs – Amazon AWS and Microsoft Azure (described in Table 6.1 as Dataset4).

We derived outage ground truth from the dataset by going through the respective timeseries, grouping measurements into probe trains and marking probe train as outage if all measurements therein exceed 5 seconds. We then evaluate the estimation power of varying probe train size – by randomly selecting $k = 1, 2, 3$ or 4 measurements from every probe train and, using a Majority voting scheme, we decide whether the train measures outage or not. ($k = 5$ was omitted as the selection of 5 measurements of a probe train of size 5 is unambiguous) We then evaluate this classifier using *precision* and *recall*:

$$\text{precision} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|} \qquad \text{recall} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|} \tag{4.10}$$

Recall is by definition perfect, as the *Majority voting* always retrieves all outages (any nonempty subset of 5 timed–out measurements contains only timed–out measurements). Precision is expressed using false outage probability in Table 4.1. Given the random nature of measurement selection, we have conducted 30–runs per every *(k, protocol, VP)* triplet and report mean μ and standard deviation σ. We can observe the positive effect of redundant measurements on false outage probability, whose analytic results were shown in Figure 4.7. A staircase–like descent of probability is caused by Majority rule that requires 2 timeouts out of 2 or 3 measurements, but 3 timeouts out of 4 measurements to flag an outage.

Table 4.1: ***False outage probability [in %] when increasing probe train size.***

	protocol	HTTP				TCP			
	# probes	1	2	3	4	1	2	3	4
AU	μ	47.41	26.62	27.44	13.20	67.53	36.74	34.18	15.85
	σ	3.93	4.65	4.88	3.30	2.59	3.57	4.25	3.83
CZ	μ	53.52	42.23	42.10	26.73	55.77	44.82	45.61	31.36
	σ	2.68	1.69	1.58	2.38	2.35	1.46	1.62	1.85
JP	μ	53.92	32.75	34.93	21.48	57.25	24.38	25.57	13.32
	σ	4.22	5.00	4.86	2.59	3.59	3.70	4.22	1.51
BR	μ	32.90	33.30	33.19	26.05	28.92	29.13	29.18	22.04
	σ	0.40	0.35	0.34	0.33	0.43	0.31	0.32	0.34
US	μ	47.70	9.31	9.91	4.91	48.29	10.32	10.86	5.70
	σ	1.89	1.97	1.28	0.43	2.10	1.53	1.75	0.61

We can also observe that HTTP and TCP descend simultaneously, but sometimes differ significantly from either side. That confirms both the underestimation and overestimation risk of *active–probing* techniques, as well as extra information gained by probing multiple protocol layers.

4.4 Conclusion

We have formally described the variables of the multidimensional Cloud latency monitoring methodology. We have evaluated the methodology and quantified the high accuracy this technique exhibits, using the accuracy gains at various measurement dimensions. However, the measurement setup must be reasonable, because high probing frequencies, high traffic volume or excess probes would perturb the Cloud service, skew measurement results and introduce extra costs. In the following Chapter, we use the aforementioned best practices to introduce the data capture platform as a method to conduct measurements and store the results thereof.

Chapter 5

Data Capture Platform

5.1 Introduction

In this Chapter, we describe the general terminology of distributed monitors and use it to classify our data capture platform called *CLAudit*. We then detail its prototype implementation and deployment, as well as the measured Cloud variables and measurement setup. Lastly, we show the dimensionality of the measurements[1] through examples.

5.2 Monitor Classification

A monitor is a tool used to observe activities on a system. In general, monitors observe the performance of systems, collect performance statistics, analyze the data, and display results. Some also identify problem areas and suggest remedies [70].

Monitors are classified based on a number of characteristics, such as the implementation level, trigger mechanism and result–displaying ability.

Depending upon the level at which a monitor is implemented, it is classified as *software*, *hardware*, *firmware* or *hybrid* monitor.

Depending upon the mechanism that triggers the monitor into action, a monitor is classified as event driven or timer driven (*sampling* monitor). An *event–driven* monitor is activated only by the occurrence of certain events. Thus, there is negligible monitoring overhead if the event is rare. But if the event is frequent, it may cause too much overhead. The sampling monitor is activated at fixed time intervals by clock interrupts. Sampling monitors are ideal for observing frequent events. On activation, the monitor records

[1] We use the terms data and measurements interchangeably

Table 5.1: ***CLAudit monitor attributes [70].***

Attribute	Value
state–changing event	network, storage and computation latency dynamics
measured domain	message RTT
logging trace	8–tuple (VP, designation, CSP, DC frontend, DC backend, protocol, RTT, timestamp)
system overhead	storage + tens–to–hundreds of lightweight spread–out network messages per few seconds
input rate	sampling rate of 10^2 per hour
input width	108 bytes
resolution	milliseconds, can measure simultaneous events

device–status registers and counters. The frequency of sampling is determined by the event frequency and the desired resolution.

Another way to classify monitors is according to their result–displaying ability. *On–line* monitors display the system state either continuously or at frequent intervals. *Batch* monitors, on the other hand, collect data that can be analyzed later using a separate programmatic analysis.

We have implemented the Multidimensional Cloud latency monitoring *methodology* through a *distributed software sampling on–line* and *batch monitor* called *CLAudit*. Its attributes are summarized in Table 5.1.

Distributed–system monitors are conveniently viewed using layers of their functions. Each layer makes use of the services provided by the lower layers and extends the available facilities to the upper layer. Here we paraphrase a reference description of each layer [70], as well as its instantiation inside the CLAudit (see Figure 5.1).

Observation layer gathers raw data about individual components of the system. Generally, each component may have an observer designed specifically for it. Thus, there may be several observers located on different subsystems.

Collection layer collects data from various observers. It is possible to have more than one collector in large systems.

Analysis layer analyzes the data gathered by various collectors. It may consist of various statistical routines to summarize the data characteristics. Simple analysis such as counting of events is conducted most efficiently in the observer and is not considered part of the analyzer.

Presentation layer deals with human user interface. It produces, for example, reports, displays and alarms.

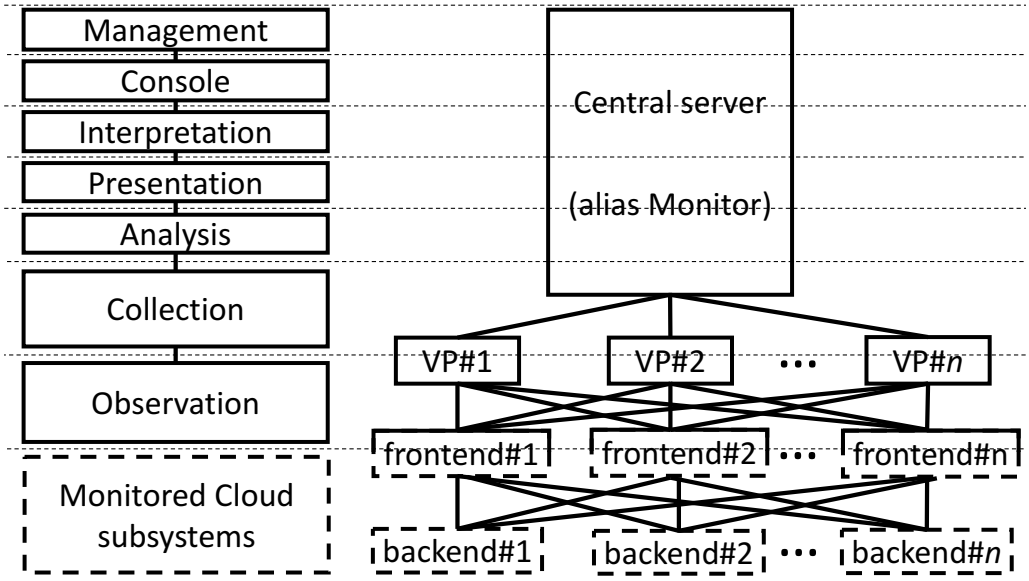


Figure 5.1: ***Correspondence of distributed–system–monitor layers (left) to CLAudit (right).*** *Every VP observes every frontend's latency and every frontend observes every backend's latency. Hierarchical data collection consists of frontends advertising to VPs, followed by a periodic solicitation by the central server. Upper layer functions are implemented on the central server.*

Interpretation layer is an intelligent entity (usually a human being or an expert system) that can make meaningful interpretations of the data. This generally requires multiple rules and trend analyses. Simple threshold–based alarms may be considered part of the presenter rather than of the interpreter, which usually requires the application of more sophisticated rules.

Console layer provides an interface to control the system parameters and states. Strictly speaking, console is not a part of the monitor. However, the monitoring and control functions are often used together and it is desirable to allow system control, as well as system observation facilities to be used together.

Management layer makes the decision to set or change system parameters or configurations based on interpretation of monitored performance. The manager implements its decision using a console. A software manager component exists only in monitors with automated monitoring and control facilities.

CLAudit components that implement the aforementioned layers of functionality (Figure 5.1) are as follows:

Vantage Points (Observation, Collection)

Vantage Points (VPs) emulate client appliances of Cloud application and service users. Via *active probing* using various real protocols, they record latencies that end users really perceive when using Cloud applications and services. To maintain comparability, VPs are

homogeneous in terms of OS and measurement software. VPs are remotely controlled and instructed to issue probe trains towards applications hosted on Frontend servers to obtain RTTs at various measured protocol layers, providing evidence for reasoning about Cloud application and service behavior. VPs are geographically dispersed to obtain end–user perspectives from around the globe. For the sake of redundancy and validation, VPs are deployed in triplets in every region (two VPs in a single campus and a third, backup VP, in different campus nearby).

Frontend servers (Observation, Monitored subsystem)
Frontends emulate client–facing servers deployed in the Cloud DCs (in our context, they respond to VP requests). They are provisioned in the exact same way as regular production–environment servers and are equipped with various real applications and services. As such, a frontend cannot distinguish between serving to end–user's client and to a CLAudit VP. Besides being among the monitored Cloud subsystems, frontends also fulfill the observer role by measuring RTTs to backend servers and advertising these to VPs. The collection function is implemented using a trap instruction inside the response–composing application code, which triggers a probing round against a predefined backend server and attaches the result to the response payload for the VP. Frontends are geographically dispersed across public CSP DCs, allowing to evaluate various real–world setups, such as a geo–distributed Cloud application or a Cloud application with redundancy.

Backend servers (Monitored subsystem)
Backends provide data when frontends need them to compose the response for client (e.g., a dynamic webpage backend or authentication service). Backends are not involved in all possible CLAudit interactions (e.g., a frontend serving a static webpage to a client). Whenever backends are queried, the overall user–perceived application latency increases as the response–composing process is often blocked at the frontend. Backends neither observe anything, nor collect measurements. They just serve as monitored Cloud subsystems. Some backends may be co–located with frontends within a single DC, whereas other backends may reside in DCs elsewhere. Backend measurements allow to evaluate multitier application setups (e.g., remote or geo–redundant data store).

Monitor (Collection, Analysis, Presentation, Interpretation, Console, Management)
The Monitor is a central server that fulfills all the monitoring functions except those of the observation layer. It is the top–of–the–hierarchy collector, as it periodically solicits

measurements from all VPs (subordinate collectors). It also performs measurement interpretations, archives past measurements, instructs VPs about measurement adjustments in a user–configurable way and visualizes near–realtime measurements and analysis results. The Monitor is to be hosted on high–end servers with failover capabilities.

5.3 Prototype

CLAudit is designed to be generally applicable, e.g., as a 3rd–party data service, a CSP monitoring solution or as SLA–verification tool for tenants. We have assembled a CLAudit prototype using readily available software tools, PlanetLab NREN [113, 28] and basic–tier subscriptions at Microsoft Azure [4] and Amazon AWS [2] public CSPs. Frontend servers are implemented as web servers and backend servers are implemented as database servers. We have implemented a subset of network communication protocols and measured their latency using several variables. We have created a representative deployment of Cloud clients and deployed comparable services at common CSP DC locations. Using a planned measurement process and the webpage–retrive scenarios, we have conducted a trial monitoring run and visualized the obtained measurement dimensionality.

5.3.1 Measurement Setup

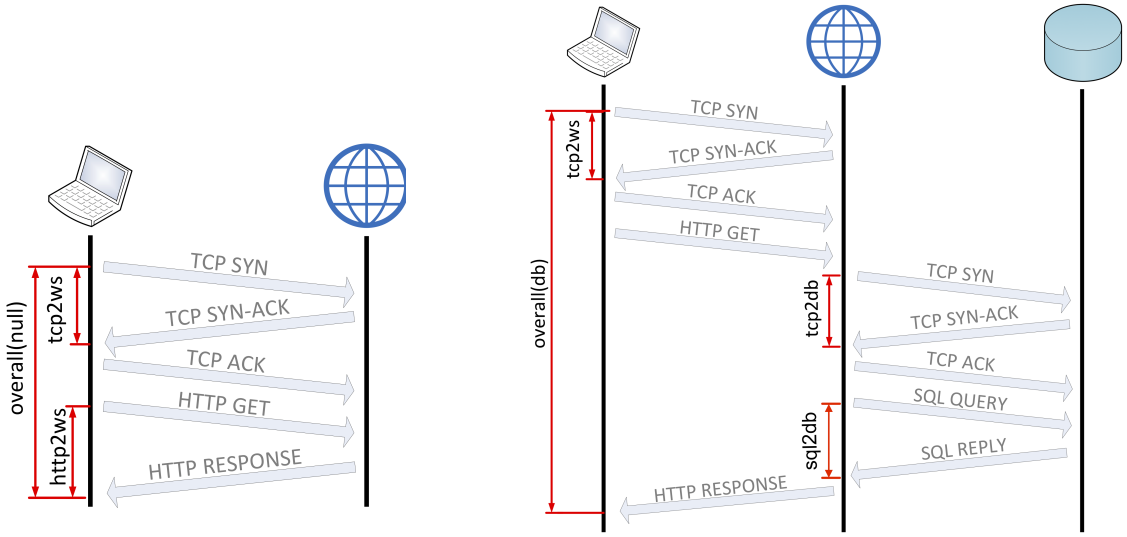
The request–response nature of many existing communication protocols allows for measuring RTT and deriving latency. CLAudit does that in a continuous, simultaneous, large–scale distributed manner. Many Cloud–related latencies, a.k.a. variables in CLAudit terminology, are measured. Table 5.2 describes variables measured in static and dynamic webpage–retrieve scenarios (shown in Figure 5.2).

The relevant variables are measured via *active probing* between every *(VP, frontend)* and *(frontend, backend)* pairs. There is neither a frontend, nor a backend selection algorithm, as *all* pairwise combinations are measured using backends predefined at VPs. CLAudit thus records Internet, intra–DC and inter–DC latencies.

Note that *tcp2ws*, *tracert* and *tcp2db* do not involve the actual request processing at server and, as such, approximate network latency well. End–to–end variables, *overall(null)* and *overall(db)*, reflect web–browser user experience. Respective protocol latencies, however, do not completely add up to these end–to–end variables, due to additional TCP/IP stack processing latencies (as seen through inter–interval gaps in Figure 5.2). These are difficult to capture externally. The secondary motivation behind *overall(null)*

Table 5.2: ***Measured variables.***

measured variable	description	backend involved
tcp2ws	Time elapsed between VP's TCP SYN sent and TCP SYN ACK from web server received	no
tracert	Time elapsed between VP's ICMP Echo Request sent and ICMP Time Exceeded from farthest reachable network hop received. In the 2013 dataset, ICMP Time Exceeded from a host containing an IATA–like airport code of target DC location was used instead	no
tcp2db	Time elapsed between frontend web server's TCP SYN sent and TCP SYN ACK from backend database server received	yes
sql2db	Time elapsed between frontend web server's SQL query sent and SQL reply from backend database server received	yes
http2ws	Time elapsed between VP's HTTP request sent and static web HTTP response from frontend web server received	no
overall(db)	Time elapsed between VP's TCP SYN sent and HTTP response from frontend web server, incorporating backend database data, received	yes
overall(null)	Time elapsed between VP's TCP SYN sent and HTTP response from frontend web server received	no



(a) *Static webpage retrieve* **(b)** *Dynamic webpage retrieve*

Figure 5.2: ***Webpage retrieve–time breakdown.*** *Static and dynamic webpage retrieves with latencies measured by CLAudit.*

and *overall(db)* is to validate the subordinate protocol measurements (e.g., TCP), via an independent measurement conducted using different software tools.

A single VP probing round consists of a train of *five* requests of every measured variable (Table 5.2) and up to five respective responses received, repeating every three or four minutes[2]. This setup and other parameters are based on findings from Section 4.3, which guides the minimal values that should be used to achieve highly–accurate dependable data. Higher values (e.g., of probing round period, probe train size, number of protocols) are not cost–efficient and would perturb the Cloud and skew measurements.

Measurements are asynchronous to avoid coordinated omission of information (i.e., not backing subsequent measurements off due to delayed or missing response to previous measurements). Measurements are not intended to be a stress test – neither PlanetLab's, nor CSP's acceptable user policy is violated. Partly because there are just a few packets generated and partly because these are spread over time due to unequal distances between VPs and DCs. Monetary cost of the experiment remains thus low.

RTTs are measured in milliseconds, rounded down to a nearest integer. Smaller resolution would increase the error due to shared and virtualized nature of PlanetLab, which introduces multiplexing delays. All requests have a reasonable five seconds timeout set, effectively treating timeouts and failures to respond the same way. The five RTTs recorded at every time instant can be viewed as five random variables. The described measurement setup allows to reveal behavioral patterns lasting tens of minutes or longer. Shorter events cannot be effectively distinguished.

5.3.2 Implementation

Various platforms, execution environments, software tools and other technologies were used to instantiate the CLAudit prototype. Figure 5.3 depicts CLAudit components along with respective execution environments that host key monitor functions and modules for inter–component interactions.

All VP nodes are PlanetLab i386 machines running Fedora Core 8 OS and are provisioned with OSS packages adjusted for measurement–probe generation and RTT–measurement purposes. A `cron` scheduler, running on the VPs, is instructed to read the YAML configuration file on every probing round. The recorded quintuple of RTTs is appended to a daily CSV log along with timestamps. Measurements are locally buffered

[2] three minutes in 2013–2015 and four minutes from 2016 on

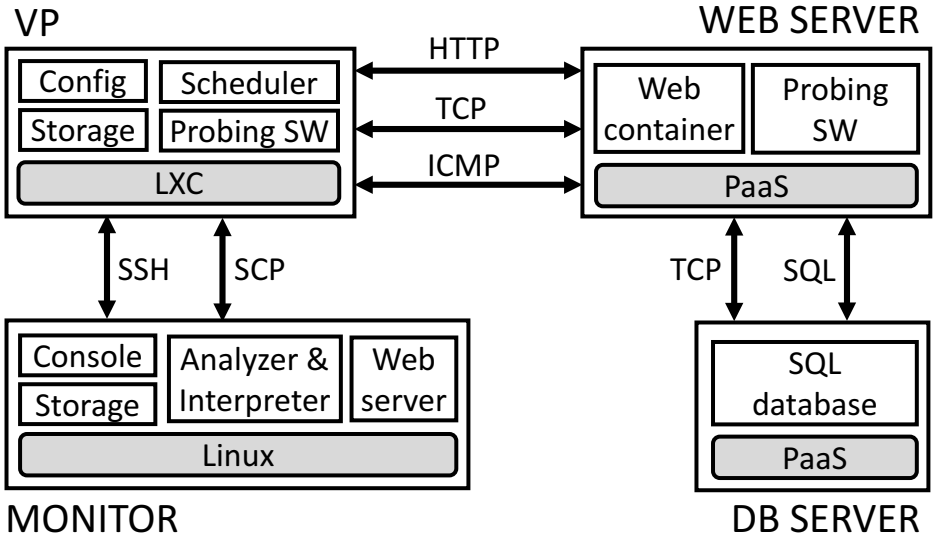


Figure 5.3: ***Component view of CLAudit.*** *Four major components along with execution environments that host functions and modules for inter–component protocol interactions.*

and downloaded by the central server every twenty minutes. Measurements are transferred uncompressed in CSV–formatted files. The actual measurement script is written in Python and calls various GNU UNIX utilities (`curl` [14], `httping` [19], `tcping` [33] and `traceroute` [35]). The implementation is modular and allows for easy extension by new probe types to mimic various Cloud applications. All VPs have their clock synchronized via the Network Time Protocol (NTP) to enable comparison of the asynchronously measured data with a negligible clock skew.

Frontend servers are implemented using basic–tier shared PaaS VMs of public CSPs. At Microsoft DCs, frontends are implemented using the Azure Web Apps. At Amazon DCs, frontends are implemented using the AWS Elastic Beanstalk. At every measured DC, a frontend consists of a single instance without backup. The measured Cloud application is a web container combining tiny static and dynamic webpages. When an HTTP request for a dynamic webpage arrives, the application is instructed to issue a database SQL query and record a database TCP and SQL RTTs into HTTP response for the VP (Figure 5.2b). The static or the dynamic webpage fits inside a single HTTP response packet. The SQL querying mechanism is implemented using the PHP PDO API and the TCP handshake is implemented using the raw TCP socket PHP API.

Backend servers are implemented using basic–tier shared PaaS VMs of public CSPs. At Microsoft DCs, backends are implemented using the Azure SQL at S1 performance level. At Amazon DCs, backends are implemented using the MySQL RDS. At

every measured DC, a backend consists of a single instance without backup. There is no data stored inside the databases, as frontends request only a random number. Backends do not interact with VPs directly, but only via frontends.

The Monitor is implemented using a single 16GB Intel x64 Ubuntu server. It interacts with the VPs via SSH channels for measurement setup adjustments and via SCP channels for measurements transfer. It stores the data downloaded from the VPs inside RAID. Advanced data analysis and interpretation are conducted online on–demand using MATLAB and Python scripts. Latency timeseries, past measurements, analysis results, alarm–mode metrics and Cloud failure monitoring traces are visualized online through CLAudit website that is implemented using JavaScript plotting libraries and PHP scripts, hosted on the Monitor's local Apache Tomcat webserver. The Console is implemented using a distributed YAML configuration file and allows for adjusting an ON/OFF switch, a probing round period and a set of measured frontends and backends. The prototype does not have monitor management facility automated and human input is thus necessary.

Additional issues had to be addressed to make measurements dependable. In the following paragraphs, we describe preprocessing tasks such as emptying HTTP caches, populating DNS caches, warming connections up, preventing database connection pooling, selecting protocol versions, avoiding ICMP message filtering and CDN RTT cutting.

HTTP is a popular protocol suitable for caching. It can reduce the number of unnecessary round trips and the overall amount of the HTTP data transferred. This blurs our measurements in the cases where we expect a certain number of round trips to occur. Thus, VP caching capabilities have been turned off. A similar problem concerns HTTP version 1.1, which uses persistent TCP connections that prevent future TCP handshakes from happening. HTTP 1.0 was used instead. CSP's large CDN infrastructure can present end users an illusion of the original resource or application being nearby, which is not acceptable in the cases when we seek data from the original Mega–DC in order to measure its network RTT.

Web applications also often maintain database connection pools. These keep the TCP channel up and, following the first successful connection, just the SQL messages are exchanged over the channel. Because we gather, among others, raw periodic TCP RTTs between the web server and the database, we force a TCP handshake, by purging connection pools. Finally, SQL data caching was prevented by always requesting random data from the database.

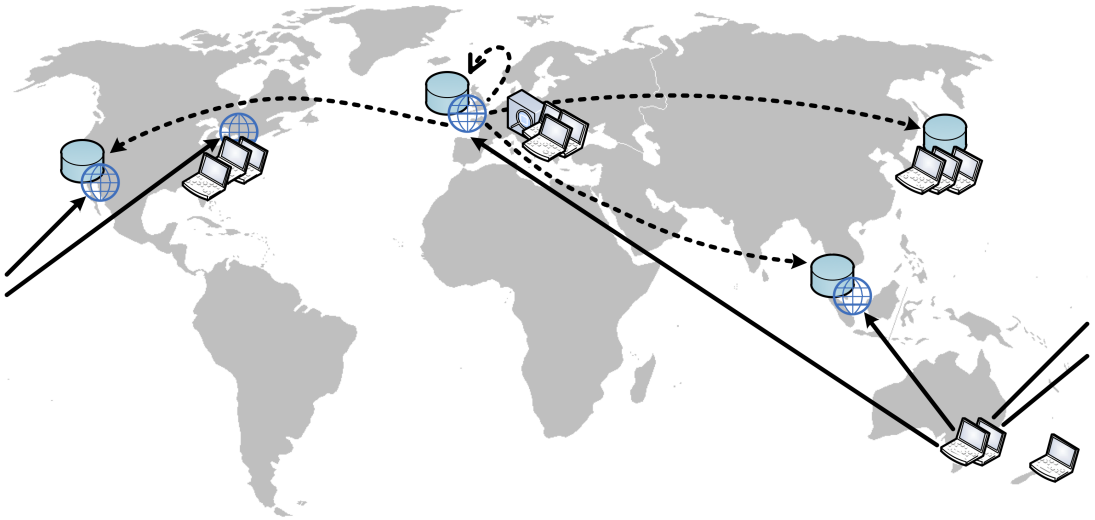


Figure 5.4: ***CLAudit deployment.*** *VPs are represented by laptops, frontend servers by globe icons and backend servers by cylinder icons. Many of the monitor functions are executed on the central server, represented by lens icon. Continuous and dashed curves depict subset of VP–to–frontend and frontend–to–backend measurements, respectively.*

5.3.3 Deployment

CLAudit monitoring was launched in 2013. Given the ever–changing set of available PlanetLab nodes, the VP deployment has been modified several times, striving to preserve comparability and ensure monitoring continuity. CLAudit prototype deployment, as of 2017, is reflected by Figure 5.4.

All CLAudit components are distributed worldwide, but not uniformly. VPs are deployed in representative locations on each continent apart from Africa. In 2017, they resided in Atlanta (USA), Prague (Czech Republic), Hiroshima (Japan) and Melbourne (Australia). VPs exist in triplets for backup purposes in the case of PlanetLab maintenance or failure event. The third VP always resides in an independent campus, different from that of the primary and secondary VPs (backup campus is in the same country though, ideally nearby). Measurements are conducted simultaneously from all three VPs. At times when one VP is not operational due to arbitrary local or campus–wide event, measurements from the backup VP are used for analysis. Other uses for multiple VPs include verification of measurements and reasoning about anomalies.

Frontend and backend servers are also distributed globally, hosted in the Microsoft Azure and Amazon AWS DCs. In 2017, frontends resided in California (USA), Virginia (USA), Singapore (Singapore) and Dublin (Ireland). Backends resided in California (USA), Tokyo (Japan), Singapore (Singapore) and Dublin (Ireland). The rationale

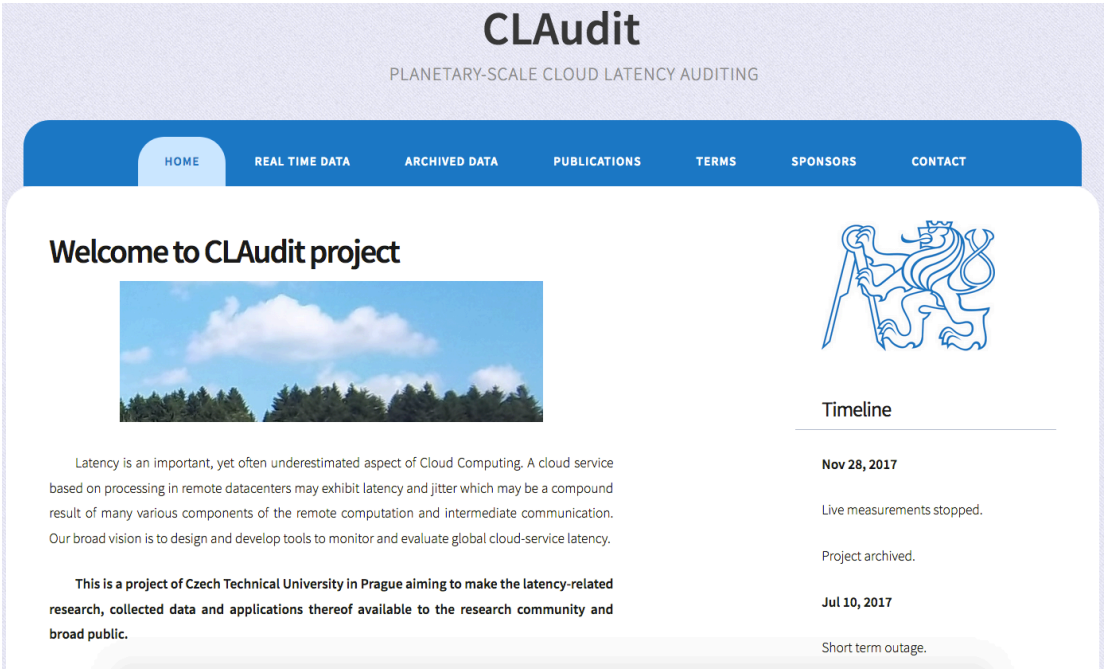


Figure 5.5: ***CLAudit landing page.***

behind the server placement is twofold. Firstly, servers have to be at common DC locations of the CSPs, to allow for provider benchmarking and other comparisons. Secondly, it has to be possible to evaluate various popular application–deployment setups (e.g., geo–redundancy, remote storage or DNS load–balanced frontend). Since CLAudit measures every pairwise combination of frontend and backend, various application–deployment alternatives are evaluated, including frontend and backend in the same DC, on the same continent or on the other side of the world.

The Monitor is hosted on premises of the Czech Technical University in Prague (CTU). The Monitor hosts the CLAudit project website (Figure 5.5), which, among others, used to visualize near–realtime measurements and still provides past–data archives. The data is available to the research community, as well as to the general public, under ODC–By license. The project website is `http://claudit.feld.cvut.cz`

5.3.4 Measurement Examples

From 0:00 UTC on April 29, 2013, we have conducted a one–week trial *CLAudit* monitoring run against Microsoft Azure California DC. This Subsection presents two examples of the data collected by CLAudit. The data represents ordinary Cloud service behavior without notable anomalies, viewed at two different dimensions of the data.

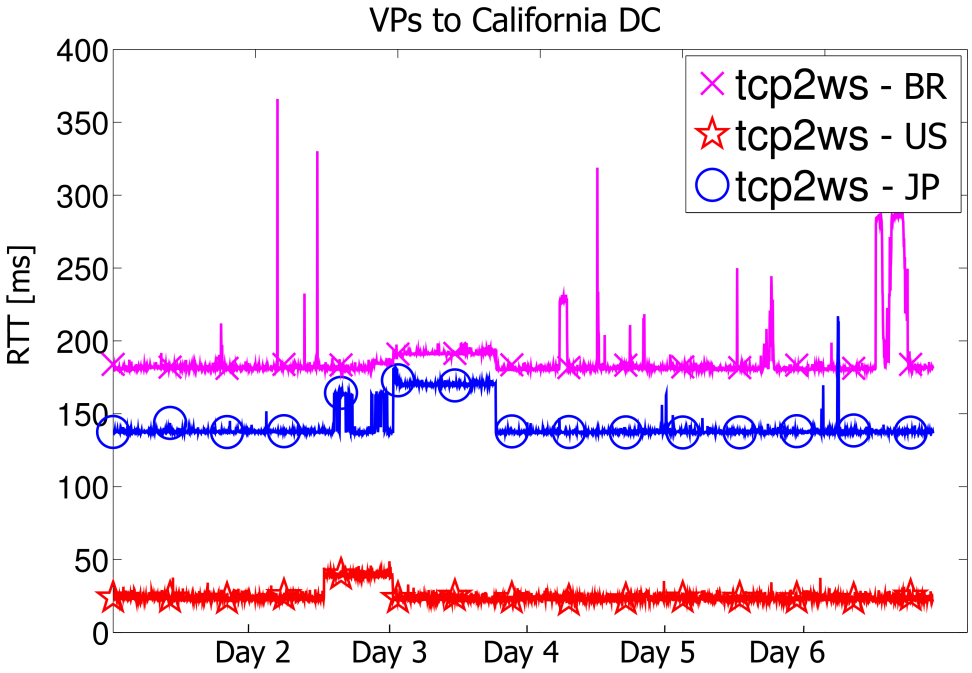


Figure 5.6: ***VP dimension example.*** *A single–week tcp2ws timeseries, as measured by VPs (Brazil, USA and Japan) TCP–handshaking with California frontend. Normal Cloud network behavior is observed.*

Vantage Point Dimension

The first example shows a *VP dimension* of the *tcp2ws* variable (Time elapsed between VP's TCP SYN sent and TCP SYN ACK from frontend server received) in the static webpage retrieve scenario. Datapoints in Figure 5.6 are median RTTs from Brazil, US and Japan VPs. Median (the 3rd largest RTT within every probe train) was used to smooth out the timeseries and filter outliers.

Latency observed by US Seattle[3] VP was very stable. It experienced low volatility and only minor spikes. The RTTs were roughly two times the optimal RTT (according to Table 6.3). The more distant VPs not surprisingly experienced more severe fluctuations and spikes. Figure 5.6 thus shows more or less normal behavior with VPs from more distant locations and inferior network infrastructures experiencing pronounced adverse network effects. Mostly observed were local effects with a couple of suspicious RTT increases simultaneously observed by pairs of VPs, increasing a likelihood of an issue with impact beyond local network.

[3]In 2013, US VP was located in Seattle

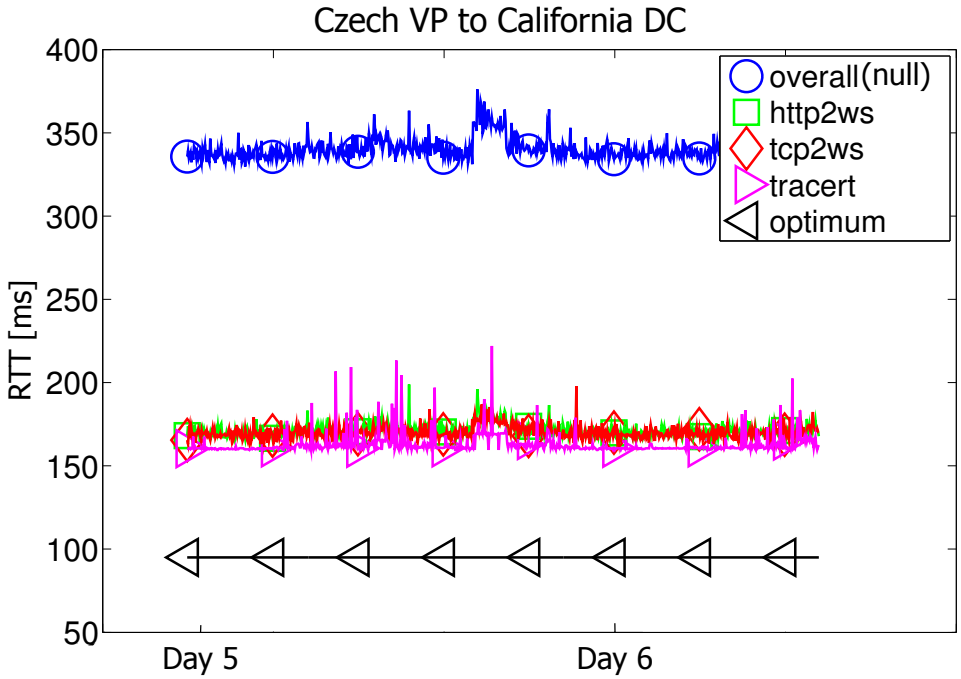


Figure 5.7: ***Protocol dimension example.*** *Roughly two days of data from various protocol layers of frontend server in California, as measured by Czech VP. The constant precalculated light's RTT denotes the optimal RTT.*

Protocol Layer Dimension

The example in Figure 5.7 shows a full *protocol–layer dimension*, i.e., all the measured variables relevant to a frontend server in the static webpage retrieve scenario, as measured during two days by the Czech VP.

Light's RTT denotes the optimum, since the lower one–way propagation delay bound is `31 ms` and signal in fiber does two round–trips at ∼66% speed of light. *tracert* RTTs are located slightly below those of *tcp2ws* and *http2ws* timeseries, because the ICMP packets did not traverse to the ultimate frontend server inside the DC (the reason being the Microsoft Azure filtering policy). *tcp2ws* and *http2ws* timeseries are almost identical, except for occasional HTTP spikes. Combined, a single round trip time of TCP followed by a single round trip time of HTTP plus a request processing delay, together approximate user–perceived end–to–end latency (denoted by the *overall* variable). That corresponds to opening of a small static webpage in the web browser for the first time (assuming no rendering delay).

5.4 Conclusion

In this Chapter, we have described a distributed–system monitor called *Cloud Latency Auditing Platform (CLAudit)*. Importantly, it serves as the multidimensional measurements–capturing platform, capable of rigorous recording of various network latencies involved in client interactions with Cloud applications and services. It can be used as a 3rd–party data service, CSP monitoring solution, SLA verification tool for tenants or research experimentation tool. We have implemented and demonstrated the representative prototype. This obviously lacks in scale and features compared to full–fledged hypothetical production implementations, but already provides representative global coverage, support for five frontend and backend protocols and facilities for monitoring configuration, latency timeseries visualization and automatic detection of suspicious events in collected measurements of target CSPs.

Chapter 6

Applications

In this Chapter, we first describe RTT datasets of major public Cloud Service Providers (CSPs), used for performance evaluation of both the Multidimensional latency measurement methodology and the data–driven application methodologies. In the following Sections, we describe and evaluate the latter methodologies that extend the state–of–the–art in three Cloud performance application areas.

Namely, in Section 6.2, we describe and analyze anomalous behaviors found in latency timeseries and then describe insights into and trends of Cloud service evolution, using *Network profiling* by selected metrics and statistical measures.

In Section 6.3, we describe measurement preprocessing and the subsequent *Benchmarking* methodology that identifies significant differences in a Cloud service performance across CSPs, their DCs and resources therein.

In Section 6.4, we formulate the *Optimization* methodology that leads to minimum adverse network effects on Cloud traffic and avoids problematic DCs. We also present an implementation using the router application plane.

6.1 Performance Evaluation Datasets

The CLAudit prototype used to collect measurements of static and dynamic webpage retrieves for almost five years. We use four subsets of the collected multidimensional measurements to experimentally evaluate our proposed data–driven applications. Specifically, we have used two comparable data blocks for deriving Cloud insights and evolutionary trends. We have used another two blocks, one for Cloud network profiling and the other one for Cloud service benchmarking and Cloud connectivity optimization

evaluation (the results based on the last block have the CSP names obfuscated). These four datasets are summarized in Table 6.1.

Over the course of CLAudit measurements, changes in a set of measured DCs, protocols and VPs were made due to various reasons. Nevertheless, preserving the comparability of the measurements was always a top priority.

An important dataset–timeframe selection criterion was the PlanetLab stability. Measurements may be disrupted for various near–end reasons, such as maintenance, overload or policy. Table 6.2 breaks down the causes of measurement disruptions, manifesting as timeouts in Dataset2 and Dataset3. The observed improvement at Dataset3 over Dataset2 is thus caused by both the availability improvements on the CSP side and by CLAudit re–engineering. The only simultaneous failure of both co–located VPs happened in 2013, between May 12 and May 13, rendering a piece of data irrecoverable and triggering the need for a third, backup VP. Industry–grade CLAudit deployments would likely use dedicated, stable and controlled machines for hosting VP functions.

CTU hosts a webpage with archived measurements (Figure 6.1). Also, from early–2013 to late–2017, this webpage was used to visualize near–realtime measurements, alarm–mode metrics, Cloud failure traces and metrics for suspicious–behavior detection. The datasets are still available to the research community, as well as to the general public, under ODC–By license. The project URL is `http://claudit.feld.cvut.cz`

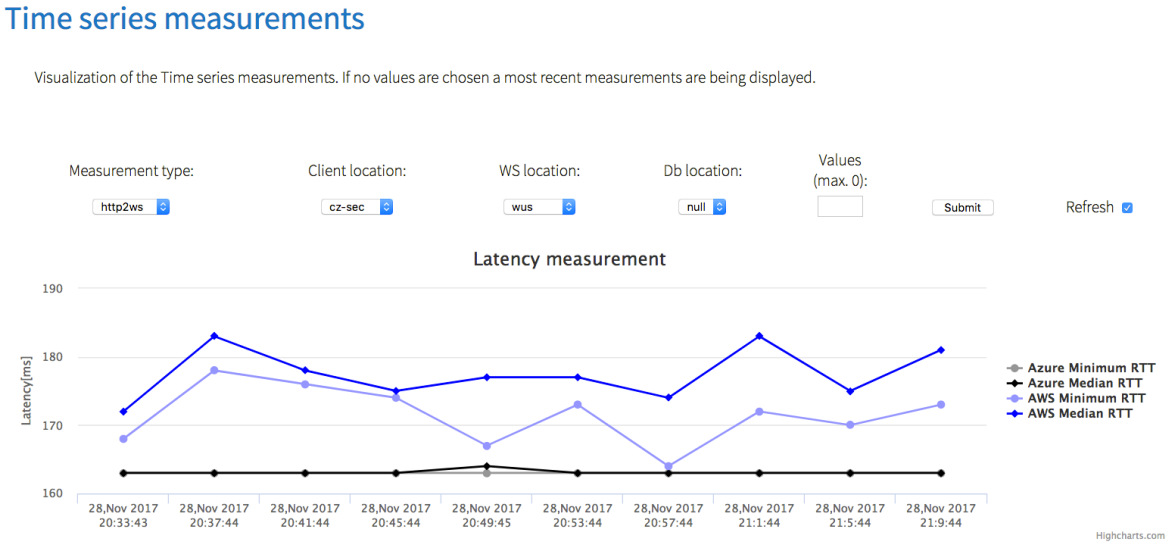


Figure 6.1: ***Online measurement visualizations.***

Table 6.1: ***Datasets.*** *Used for experimental evaluation of the monitoring and the data–driven methodologies for Cloud service improvements.*

	Dataset1	Dataset2	Dataset3	Dataset4
Year	2013	2013	2016	2016
Timeframe	Apr 29 – May 5	May 10 – Jun 4	Jan 10 – Feb 4	Jan 10 – Mar 19
Timeout threshold	5 seconds	5 seconds	5 seconds	10 seconds
Timeout visualized as	0 seconds	5 seconds	5 seconds	10 seconds
Probing round period	3 minutes	3 minutes	4 minutes	4 minutes
Probe train size	5 probes	5 probes	5 probes	5 probes
Protocols	IP, TCP, HTTP, SQL	IP, TCP, HTTP, SQL	IP, TCP, HTTP, SQL	TCP, HTTP, SQL
VPs	Australia, Czech Rep. Japan, Brazil, USA, Russia	Australia, Czech Rep. Japan, Brazil, USA, Russia	Australia, Czech Rep. Japan, Brazil, USA	Australia, Czech Rep. Japan, USA
CSPs	MS Azure	MS Azure	MS Azure	Amazon AWS, MS Azure
Frontends	California, Dublin Chicago, Hong Kong	California, Dublin Chicago, Hong Kong	California, Dublin	California, Dublin Singapore, Virginia
Backends	California, Dublin Singapore, Hong Kong	California, Dublin Singapore, Hong Kong	California, Dublin Singapore	California, Dublin Singapore, Tokyo
CSP name obfuscation	no	no	no	yes

Table 6.2: ***Fraction of timeouts.*** *Quantification of* `5000 ms` *samples inside Dataset2 and Dataset3, together with causes and their breakdown.*

	Dataset2					Dataset3				
Fraction	6.97% (66128 samples)					0.16% (1143 samples)				
Cause	PlanetLab	Other				PlanetLab	Other			
Fraction	63.28%	36.72%				24.85%	75.15%			
Backend DC	-	None	California	Dublin	Singapore	-	None	California	Dublin	Singapore
Fraction of Other	-	35.60%	0.55%	0.27%	0.29%	-	35.26%	11.99%	12.69%	15.22%

6.2 Cloud Network Profiling

6.2.1 Introduction

In this Section, we present a detailed analysis of the Microsoft Azure measurements, including notable anomalous behaviors, and a comparison of Cloud–service latency between two periods (i.e., two comparable blocks of CLAudit measurements, collected in 2013 and 2016).

6.2.2 Anomalies

CLAudit measurements, among others, allow to reveal certain anomalies and provide background for their explanation. Here we present three examples of such anomalies, discovered manually inside Dataset1 (Table 6.1) and not reported by Microsoft Azure CSP. We then analyze and reason about them. Such anomalies are not unusual, but also not necessarily inevitable. We believe that suitable monitoring and analysis can help decrease the occurrence of such anomalies and make the Cloud services more predictable.

Persistent HTTP hike

This anomaly concerns the Illinois DC, specifically the web server hosting our web container. Figure 6.2 visualizes the timeseries of interest. The incident began after 6 p.m. UTC on a weekday. It took almost 50 minutes for the server to fully recover and continue serving clients with low stable RTT.

The incident was perceived by all VPs. HTTP latency increased by an order of magnitude – RTTs, previously ranging from 50 to 500 milliseconds, increased to over 5 seconds for over 25 minutes. RTTs then started to decrease simultaneously and returned to the baseline after another 25–minute recovery period. This hike was not reflected in the TCP RTTs, measured against the same web server. That effectively points to a VM/web server issue as a likely cause. DC mechanisms such as TCP splitting or interception might prevented VP TCP connections from reaching the same ultimate target as HTTP, but from the point of view of the VPs, the transport layer performance was intact, whereas application layer performance was heavily impaired. Interestingly, this anomaly occurred almost simultaneously also in the Dublin DC, which might point to additional causes (e.g., VM migrations or various types of attacks).

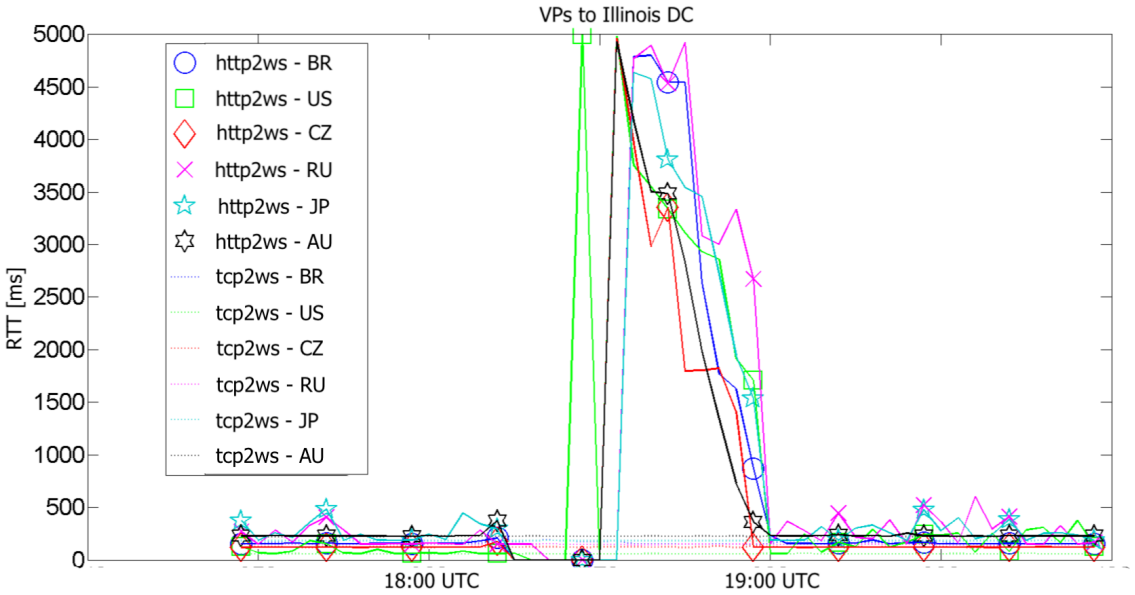


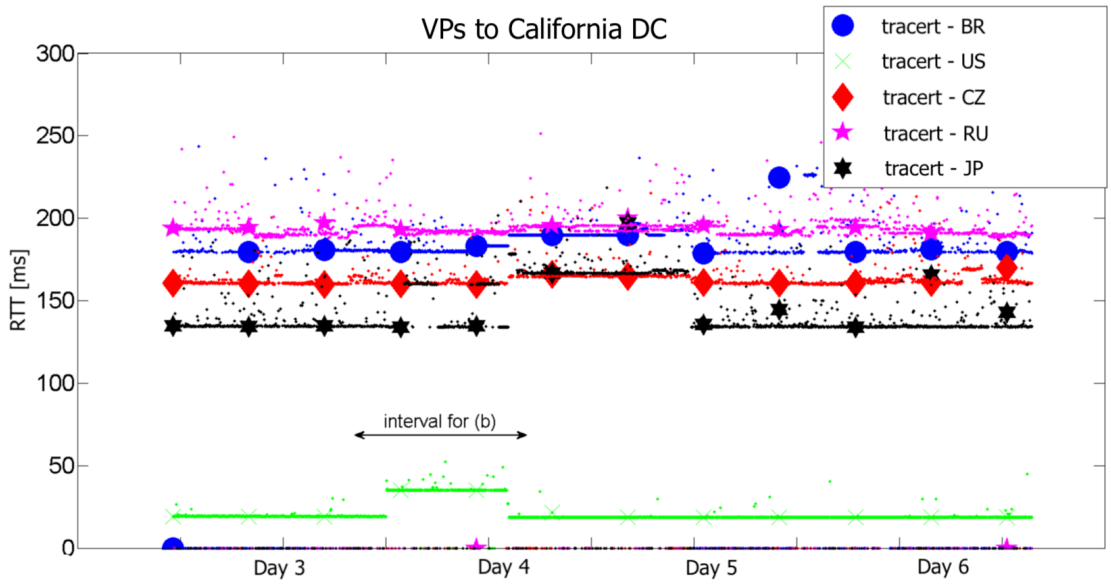
Figure 6.2: ***HTTP RTT hike****. An order of magnitude increase of HTTP RTT followed by a slow recovery, as perceived by all CLAudit VPs worldwide. The TCP RTTs targeting the identical web server remained stable, as shown at the bottom. Timed–out measurements as visualized as* `0 ms` *samples.*

Path Issue

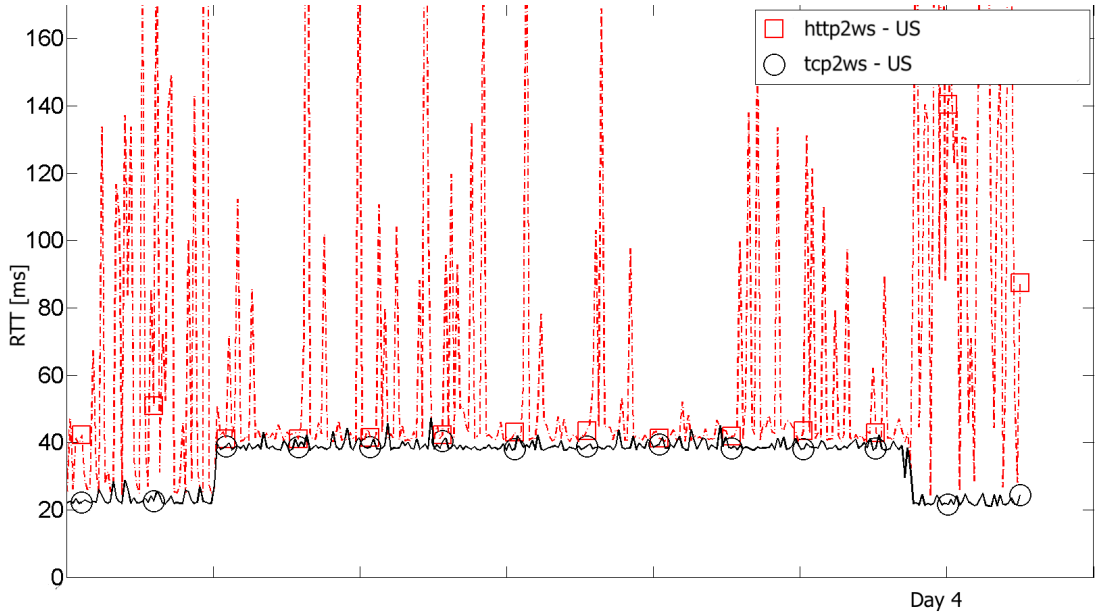
This anomaly concerns the California DC, specifically the web server hosting our web container. Figure 6.3a visualizes relevant network–layer and Figure 6.3b relevant transport and application–layer timeseries. A combination of TCP, HTTP and ICMP measurements confirmed the problem's wide span and, thus multiple Cloud applications were affected simultaneously. The anomaly was detected by CLAudit on Saturday after 12 p.m. UTC and disappeared entirely on Monday around 12 a.m. UTC.

This anomaly was first perceived by the US VP through persistent, almost doubled ICMP, TCP and HTTP RTTs. This state suddenly disappeared around 12 a.m. UTC on Sunday. Shortly after it disappeared, the situation has repeated for the other VPs (Brazil, Czech Republic, Russia and Japan). The RTT increase was most intensively perceived by the Japan VPs. The Czech, Brazil and Russia VPs experienced lower, but still notable RTT increase. The situation suddenly disappeared before Monday 12 a.m. UTC and has not repeated itself again during Dataset1 observation period, ruling out regular weekly events.

Geographical VP distribution exhibited no temporal relation to the anomaly occurrence. Symptoms point to routing manipulations and traffic diversions, possibly



(a) *Global network–layer RTT increases*



(b) *Upper–layer degradations, as observed by US VP*

Figure 6.3: ***Path issue****. Persistent RTT increase at California DC, perceived by the US VP and followed by a simultaneous persistent RTT increase perceived by all other VPs. The measurements suggest routing manipulation as a possible explanation. Timed–out measurements are visualized as* `0 ms` *samples.*

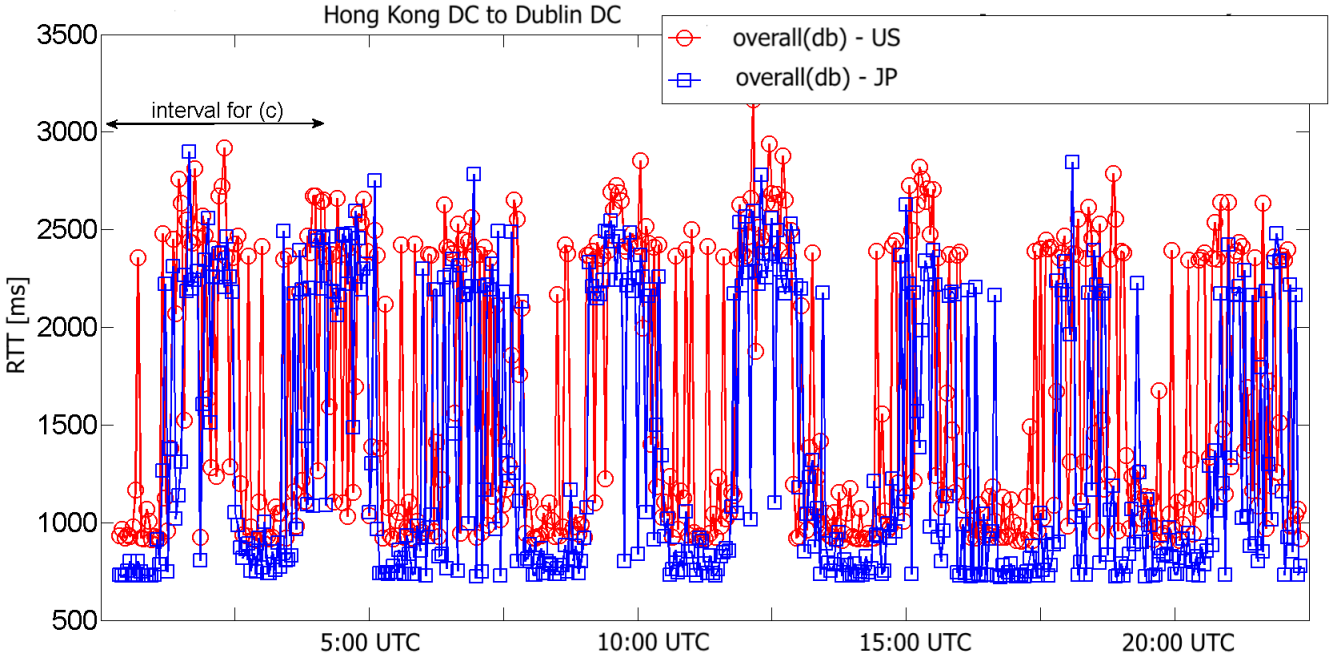
related to the weekend period. Whether intentional or not, this anomaly persistently degraded user–perceived latency of a number of applications, especially the ones with demand peaking on weekends.

Periodic Fluctuation

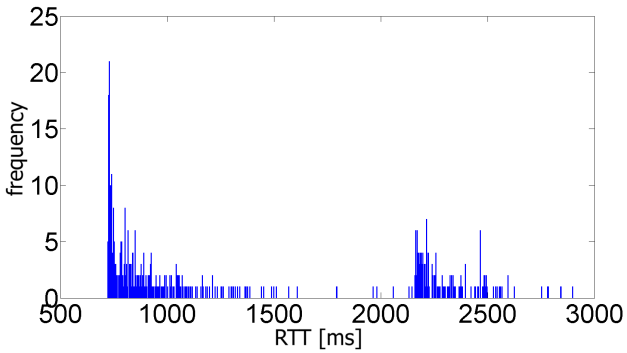
This anomaly concerns a geo–distributed application, specifically the connection between Azure frontends and backends. Figure 6.4 visualizes relevant timeseries and the deeper analysis of connection between Hong Kong DC and Dublin DC. Visualized in Figure 6.4a is the Weekday 12 a.m. to 11 p.m. UTC of *overall(db)* timeseries (*overall(db)* variable combines processing latencies, two different TCP round trips, one HTTP round trip and one SQL round trip, as shown in Figure 5.2). The measurements exhibited predictable periodic characteristics, persistent throughout the entire observation period. RTTs were eclectically shifting every 145 minutes, between periods of low stable latency and periods of high volatile latency.

Figure 6.4b shows the RTT histogram of Japan VP, with a notable distance between modes and a significant gap between the clusters. Specifically, modes are located around `725 ms` and `2200 ms`. Both parts of the distribution are right–skewed and the values that concentrate above the modes form a tail. The multitier–application user thus experienced two very different service qualities in a regularly–alternating fashion.

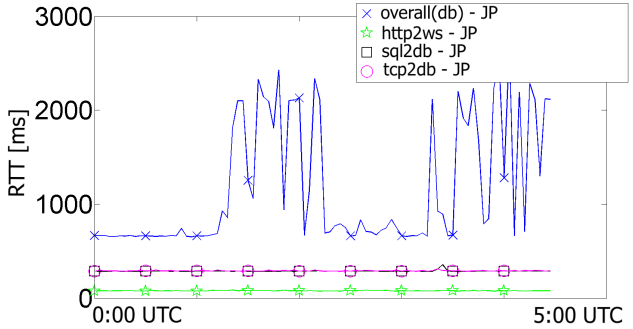
To some extent, this periodic effect was observed with all other pairwise DC combinations. Deeper analysis in Figure 6.4c reveals that the degraded periods are induced by the interbehavior of HTTP and SQL protocols. The protocol breakdown in this Figure shows no signs of fluctuations when TCP or SQL requests are sent in isolation, or when a static HTTP retrieve takes place. Determining the true origin of the problem is beyond CLAudit's capabilities at that time and necessitates a deeper diagnostic of SQL and HTTP interbehavior on Azure servers.



(a) *Periodic latency (US and Japan VPs retrieving dynamic webpage from Hong Kong web server.)*



(b) *RTT histogram (JP)*



(c) *Protocol breakdown*

Figure 6.4: ***Periodic latency anomaly of multitier geo–distributed application.*** *Alternating low stable and high volatile periods of user–perceived latency of multitier web application.*

6.2.3 Insights

This Subsection presents Microsoft Azure Cloud insights derived from Dataset2 and Dataset3 (Table 6.1). Included are *average latency per kilometer* metric, DC and protocol *availability*, latency *tail* and *multimodality*, together with notable trends over time. All these are subject to the fundamental *Speed of light barrier*, whose imposed minimum–latency lower bounds are summarized in Table 6.3

Average Latency per Distance

Table 6.4 summarizes the Dataset2 and Dataset3 using 50^{th} percentile, selected because of its robustness. To assess the Cloud–path latency in particular regions of the world, we define a universal metric called *average latency per kilometer* (Equation 6.1), which amounts to the latency accumulated by packet's propagation over one kilometer distance and, as such, ρ is an indicator of a Cloud path performance in a given region.

$$\rho(\text{src,dest}) = \frac{\breve{X}}{2.\text{dist(src,dest)}} \tag{6.1}$$

$\breve{X}$ is any kind of data summary, in this analysis we use median. The more VPs in the DC region, the greater the confidence. The optimal ρ is approximately 5.05 $\mu s.m^{-1}$. Using that value as a reference, distance from optimum can be easily seen from the actual measured ρs in Table 6.4.

The best achieved latency was a little over that of the light and was observed on the path between Australia and California. The likely explanations include, first, both sites being situated on the opposite coast of ocean where the data paths encounter fewer network hops and, second, the underseas cables approximating the *Great–circle distance* well [90]. The worst ρs (over three times the light's) were observed on the intracontinental paths, where the data paths include many hops and, because of terrain and borders, are rarely conducted along the path suggested by the Great circle.

ρ has mostly decreased between 2013 and 2016, resulting in lower–latency DC connections. Locating the particular improved network segments is difficult and would require a new set of measurements within partitioned paths. Hence, it is not clear whom the bits of improvement should be attributed to, be it the CSP, the ISP or the PlanetLab.

The consistent decrease in median latency, observed by all–but–Brazil VPs at all protocol layers, indicates an improvement in Cloud connectivity in general. Brazil VP observed an >20 ms increase for all DC frontends and protocol layers. This multidimensional observation indicates a degradation somewhere between the Belo Horizonte access connection and Microsoft's South American connection in between 2013 and 2016.

Since Azure DCs do not to let `traceroute` ICMP packets in, these packets bounce off the DC edge. This coincidentally allows us to dissect the latency improvements made inside the DC from those made outside. By subtracting median ICMP latency improvements from median HTTP/TCP latency improvements and comparing the results, we observed a decrease in both the HTTP and the TCP latency across all DCs, implying a better responsiveness of, e.g., web applications. Improvements in both the DC connection and the communication protocol latencies indicate that Microsoft is paying attention to the latency and is evolving its Azure Cloud Computing infrastructure with latency among its priorities.

Table 6.3: ***Distances and RTT lower bounds.*** *VP–to–DC distances and RTTs the light in the real–environment achieves.*

		Data Center			
		California		Dublin	
		dist [km]	RTT [ms]	dist [km]	RTT [ms]
Vantage Point	Melbourne	12644	127.81	17214	174.00
	Belo Horizonte	10404	105.16	8869	89.65
	Prague	9395	94.96	1470	14.86
	Osaka	8662	87.56	9579	96.82
	Hiroshima	8844	89.40	9498	96.01
	Seattle	1094	11.06	7300	73.79
	Atlanta	3440	34.77	6322	63.91

Table 6.4: ***Summarized RTTs.*** *Cell rows correspond to the HTTP, TCP and ICMP (going from the top). Cell columns correspond to the median of 2013 minima, median of 2016 minima, median of 2013 medians and median of 2016 medians, respectively.*

			Data Center $\tilde{X}$ [ms] (ρ [μs.m^{-1}])							
			California				Dublin			
			min13	min16	med13	med16	min13	min16	med13	med16
Vantage Point	Melbourne	HTTP	174 (6.88)	163 (6.45)	176 (6.96)	164 (6.49)	311 (9.03)	296 (8.60)	313 (9.09)	297 (8.63)
		TCP	171 (6.76)	161 (6.37)	172 (6.80)	162 (6.41)	308 (8.95)	295 (8.57)	309 (8.98)	296 (8.60)
		ICMP	169 (6.68)	160 (6.33)	169 (6.68)	161 (6.37)	-	294 (8.54)	-	294 (8.54)
	Belo Horizonte	HTTP	183 (8.79)	207 (9.95)	185 (8.89)	209 (10.04)	231 (13.02)	256 (14.43)	233 (13.14)	257 (14.49)
		TCP	180 (8.65)	205 (9.85)	181 (8.70)	207 (9.95)	228 (12.85)	254 (14.32)	229 (12.91)	255 (14.38)
		ICMP	178 (8.55)	204 (9.80)	179 (8.60)	206 (9.90)	227 (12.80)	254 (14.32)	-	255 (14.38)
	Prague	HTTP	166 (8.83)	163 (8.67)	169 (8.99)	164 (8.73)	40 (13.61)	37 (12.59)	42 (14.29)	37 (12.59)
		TCP	163 (8.67)	159 (8.46)	168 (8.94)	161 (8.57)	38 (12.93)	34 (11.56)	42 (14.29)	34 (11.56)
		ICMP	160 (8.52)	158 (8.41)	160 (8.52)	159 (8.46)	34 (11.56)	33 (11.22)	-	34 (11.56)
	Osaka (2013) Hiroshima (2016)	HTTP	140 (8.08)	135 (7.63)	145 (8.37)	136 (7.69)	278 (14.51)	252 (13.27)	329 (17.17)	266 (14.00)
		TCP	136 (7.85)	132 (7.46)	137 (7.91)	132 (7.46)	272 (14.20)	249 (13.11)	274 (14.30)	262 (13.79)
		ICMP	133 (7.68)	132 (7.46)	134 (7.73)	133 (7.52)	-	270 (14.21)	-	271 (14.27)
	Seattle (2013) Atlanta (2016)	HTTP	24 (10.97)	54 (7.85)	28 (12.80)	54 (7.85)	160 (10.96)	99 (7.83)	228 (15.62)	99 (7.83)
		TCP	21 (9.60)	51 (7.41)	23 (10.51)	51 (7.41)	155 (10.62)	97 (7.67)	157 (10.75)	97 (7.67)
		ICMP	18 (8.23)	50 (7.27)	18 (8.23)	50 (7.27)	152 (10.41)	95 (7.51)	-	96 (7.59)

Tail Latency

Cloud–scale infrastructures can ignore neither the tail latency, nor the latency variability. Our analysis revealed the presence of both in 2013 and also in 2016. The exact sources of latency variability are hard to identify, whereas sources of tail latency include network sources like congestion and application sources like power optimizations. We attribute to the CSP only those observed by multiple VPs, even though it is certainly possible for the CSP to influence a good deal of latency tail and variability introduced outside DCs (e.g., by improving exterior routing and BGP peering at strategic IXPs).

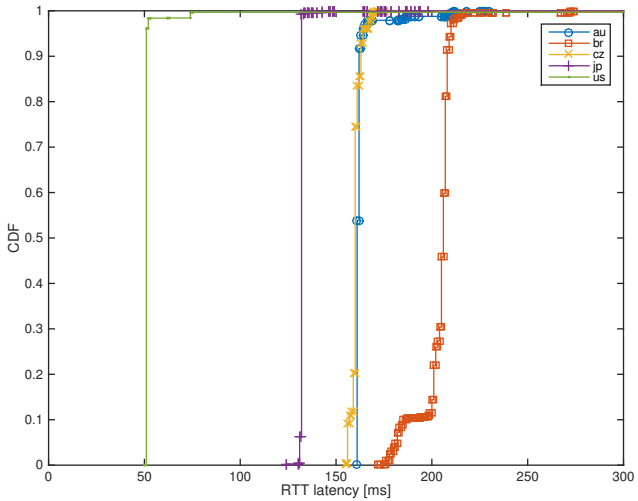
A summary of TCP CDFs is presented in Figure 6.5. Tails show up at various percentiles and sometimes add tens of milliseconds to about ten percent of the traffic, which is beyond tolerance for certain classes of applications (Figure 2.5).

Australia and New Zealand VPs observed a lot of variability and, thus have minimum and median latency CDFs apart for the most part (Figure 6.6). At Brazil and Japan VPs, variability is observed through irregular CDF shapes.

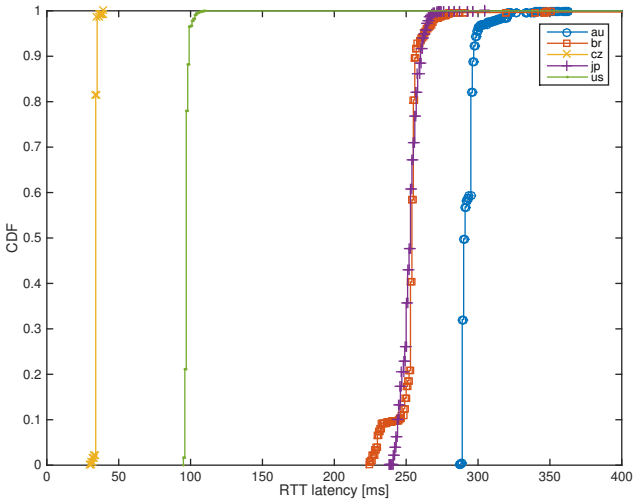
Multimodal Distribution

A common reason for discrepancies and significant jumps in CDFs (Figure 6.5) is multimodal distribution of latency, which negatively affects Cloud applications. Analysis of the timeseries identified three situations that cause bimodal distribution – a persistent increase of latency (VP–to–frontend measurements), periodic effect (frontend–to–backend measurements) and minimum–median discrepancy (VP–to–frontend measurements).

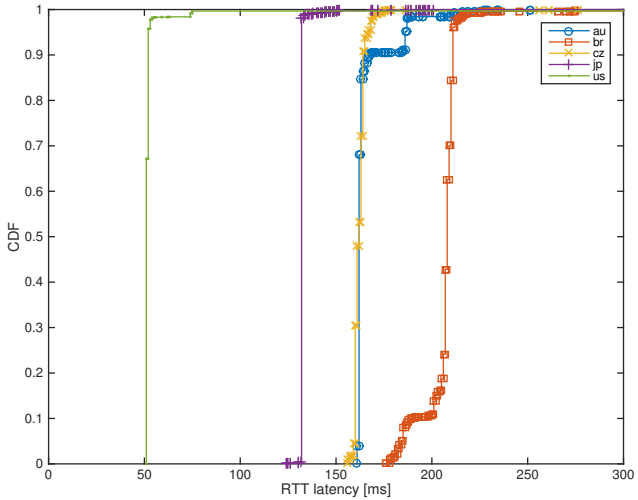
The first two situations were described as anomalies in Subsection 6.2.2. An example of the third situation is presented in Figure 6.6. Both the primary and secondary VPs (hosted in Melbourne), as well as the backup VP (hosted in Auckland), observed bimodal latency at all protocol layers when accessing Dublin DC services. As this concerns minimum and median measurements combined, it can be viewed as a bimodal mixture of two distributions, each having a different mode. The problem is likely on the path(s) from Australia/New Zealand to Dublin DC, since no other VPs observed this effect (not even on Australia/New Zealand path to California DC). Among possible causes is taking alternative paths of different latency. Alternative path does not necessarily have more network hops, as, in general, there is no strong correspondence of hopcount to distance or latency [74]. That opens room for application causes such as I/O levels, GC/hypervisor pauses, context switches, interrupts, database reindexing or cache flushes.



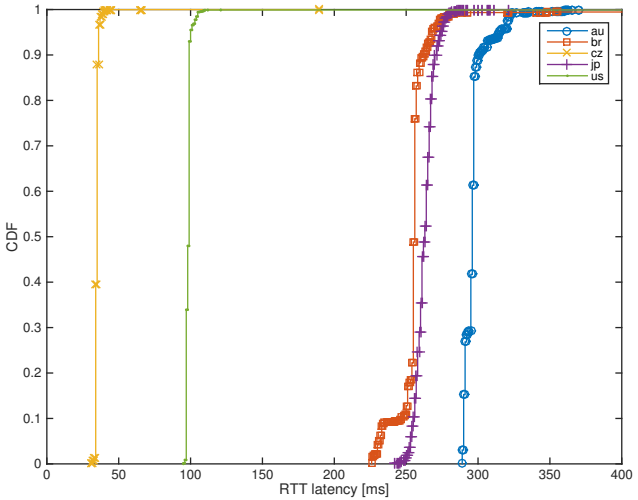
(a) *Minimum – California frontend*



(b) *Minimum – Dublin frontend*

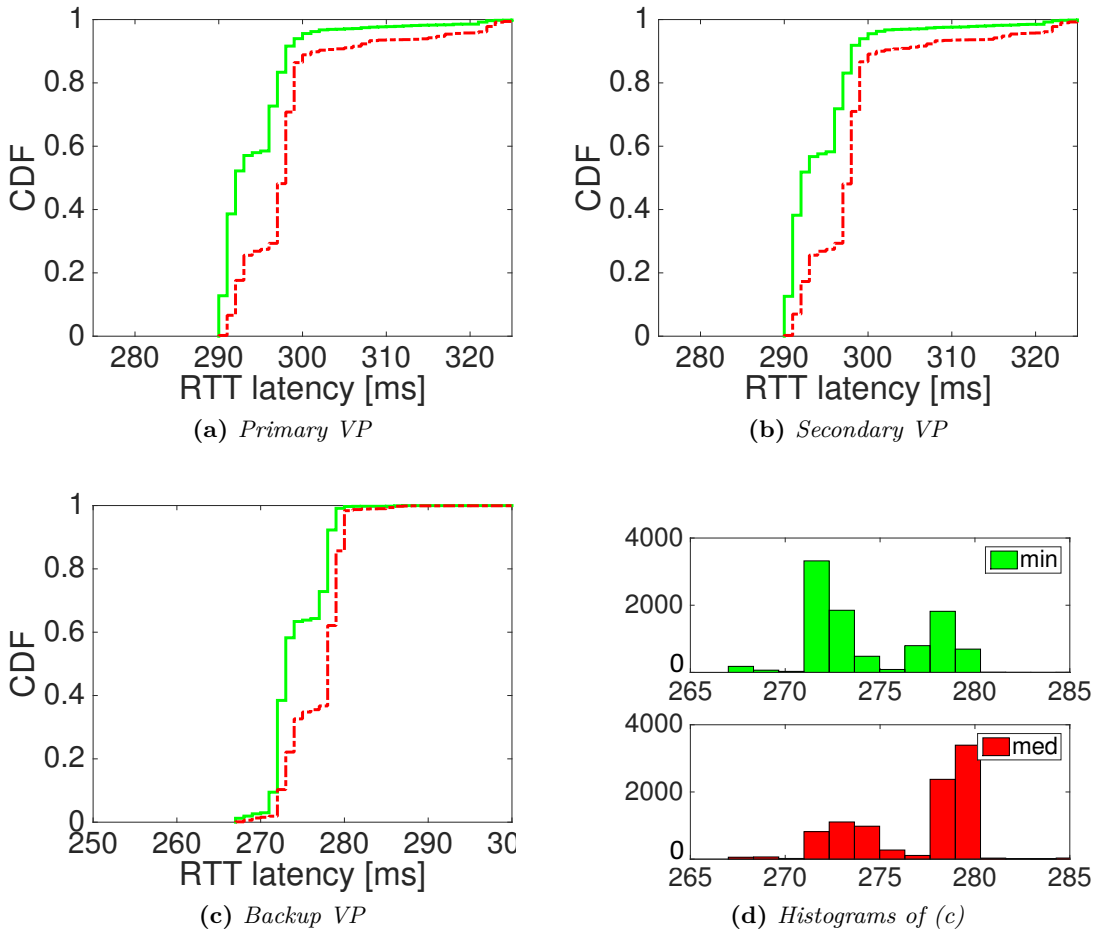


(c) *Median – California frontend*



(d) *Median – Dublin frontend*

Figure 6.5: ***CDFs of minimum and median TCP RTTs.*** *As measured between each* (VP, DC) *pair. Bimodality, tails, minor effects and discrepancies between related supposedly–identical curves are observed. Note that US and Japan VPs were relocated between 2013 and 2016 due to PlanetLab instabilities.*



(a) *Primary VP* **(b)** *Secondary VP* **(c)** *Backup VP* **(d)** *Histograms of (c)*

Figure 6.6: ***Example of a bimodal latency distribution.*** *Minimum and median HTTP RTTs apart for the most part, as observed by Australia and New Zealand VPs when measuring Dublin DC. Bimodality was observed at all protocol layers.*

Availability

DC availability and its changes over time are other important metrics. Here we consider an individual measurement successful should any of the five probes within a single train succeed (*Unanimous negative voting*). The success rate than denotes the availability. We use only backend data for availability analysis, as frontend data is heavily influenced by PlanetLab instabilities.

The results are listed in Table 6.5. Observed are availability improvements across all measured backends. Current availability tops the 99.9% mark for individual DCs, as well as cumulatively for the TCP protocol layer.

Table 6.5: ***Availability of Azure backend.*** *For TCP and per DC.*

Variable	2013	2016	Backend	2013	2016
tcp2db	99.37%	99.96%	California	99.74%	99.94%
			Dublin	99.88%	99.95%
			Singapore	99.85%	99.94%

6.2.4 Conclusion

Results in this Section provide insights into Microsoft Azure Cloud–service latency and its trends over three years. Notable observations are as follows:

- Various anomalous behaviors exist that have adverse effect on Cloud–service and application performance, often impairing user experience;
- TCP/IP protocol interbehavior is less coherent in the Cloud environment (Figure 2.8);
- Tail latency, latency variability and latency multimodality are still present on the end–to–end Cloud paths and partly explain why Cloud is not a predictable platform for all applications yet;
- Cloud backend displays a good availability over time, both protocol–wise and DC–wise.

The presented insights into VP–to–DC latency are, to some extent, explicable by the already well–researched Internet latency. Our analysis and evaluation show improvements in many aspects of Cloud–service latency over time. The impact of infrastructure improvements on the CSP side is clearly observable, most notably via the improved performance of its core networking. More results (e.g., decrease in DC–centered global–impact incidents) and an expert system for detection and interpretation of suspicious events are part of the follow–up works [120, 99].

The multidimensional measurements and analyses can be utilized for improving Cloud services and justification of offload infrastructures such as Edge Computing or Cloudlets. The multidimensional latency analysis may also lead to better geographic resource allocation on the side of Cloud–service tenants.

Furthermore, advanced data–mining techniques may be utilized to deepen the insights compared to the presented simple methods of descriptive statistics. The presented timeseries can be used to train variety of ML techniques for detection of a variety of suspicious events based on coincidence across measurement dimensions.

6.3 Cloud Service Benchmarking

6.3.1 Introduction

In this Section, we present the *Latency–based benchmarking* methodology, based on multidimensional latency measurements. We also describe measurements preprocessing, present a longitudinal case study and experimentally evaluate all methods using a large open CLAudit dataset.

With CSP offerings homogenizing, competitive differentiation takes place at a service quality level. However, CSPs reveal only insufficient amount of technical information about their service, often leaving tenants indecisive. Using Latency–based benchmarking, we show that there are significant differences in service quality among CSPs and even among single–CSP's DCs and resources therein.
Latency–based benchmarking also allows for:

- Cloud service comparisons using diverse application requirements;
- Identifying a best–fit CSP/DC for a global client base;
- Identifying a best–fit backend for a given frontend;
- Usage of in–house collected or 3$^{\text{rd}}$–party data;
- Performance estimations without actual deployments;
- Updatability of the results as new measurements get collected;
- Addition to the existing benchmarking suites.

Pursuant to the benchmarking best practices, we obfuscate and randomize the order of CSPs – Amazon AWS and Microsoft Azure. From now on, we thus use P1 and P2 to denote our compared CSPs.

Table 6.6: ***Summary of benchmarking notation.***

Term	Definition
S	Set of measurement sources, i.e., VPs and Cloud frontends
D	Set of measurement targets, i.e., Cloud frontends and Cloud backends
M	Set of RTT–summary statistics
L	Set of protocol layers of network communication stack
V	Set of Internet Vantage Point locations
F	Set of frontend–resource DC locations
B	Set of backend–resource DC locations
P	Set of Cloud Service Providers
x_n^*	RTT measurement with parameters * at n–th time instant
$\vec{v}_m^*$	Vector of m–th summary–statistic's values over all sources
w_m	Weight of summary statistic m
$r_{d,p}$	Benchmark score of CSP p's DC d

6.3.2 Data Preprocessing

We have used Dataset4 (Table 6.1) for the purpose of Latency–based benchmarking design and experimentation. As per terminology in Chapter 4, a monitoring dataset X consists of timeseries $\vec{x}$ that contain N single multidimensional RTT measurements x_n, where every successive k measurements belong to a probe train captured around a common time instant. By considering only median measurement (3^{rd} highest RTT of a sorted probe train of size $k = 5$) and VP designation $q = primary$, a measurement x_n is described as follows:

$$X^{L,V,F,B,P} = \{x_n^{l,v,f,b,p}\},\ n \in [1, 2, \ldots, N] \tag{6.2}$$

The notation used in this Section is summarized in Table 6.6.

The actual recorded latency values (visualized in Figures 6.7a and 6.7b) largely correspond to geographical distances between the sites, resulting in varying scale that makes any immediate comparisons difficult. We thus normalize the values to a new standard range. Two approaches are feasible for selection of the baseline used for normalization of raw measurements:

Normalization via optimal–latency approximation

This approach normalizes RTTs using an approximation of the optimal signal propagation latency between the measurement source and destination. Path length is approximated by the Great–circle distance (GCD). It then gets divided by the speed of

light in fiber. The optimal RTT is twice the resulting one–way latency:

$$\mathrm{RTT}_{\mathrm{optimal}}(\mathrm{src},\, \mathrm{dest}) = \frac{2.\mathrm{gcd}(\mathrm{src},\, \mathrm{dest})}{c_{\mathrm{fiber}}} \tag{6.3}$$

There are two downsides to this normalization. First, it might prioritize long distances over short ones. In the case of long distances, fiber is usually employed (e.g., transoceanic submarine cables with few hops), as opposed to short distances, where copper is common and relative number of network hops is higher. This might result in bias.

Second, based on different experiments, resulting values are poorly normalized (mainly clustered around 150%, 170%, and 190% of the ideal value), which is not suitable for further analysis. We thus refrained from normalizing using an optimal propagation latency.

Normalization via minimum RTT

This approach normalizes the values to a new standard range using minimum RTT, i.e., a minimum measured latency across all CSPs (Figure 6.7b):

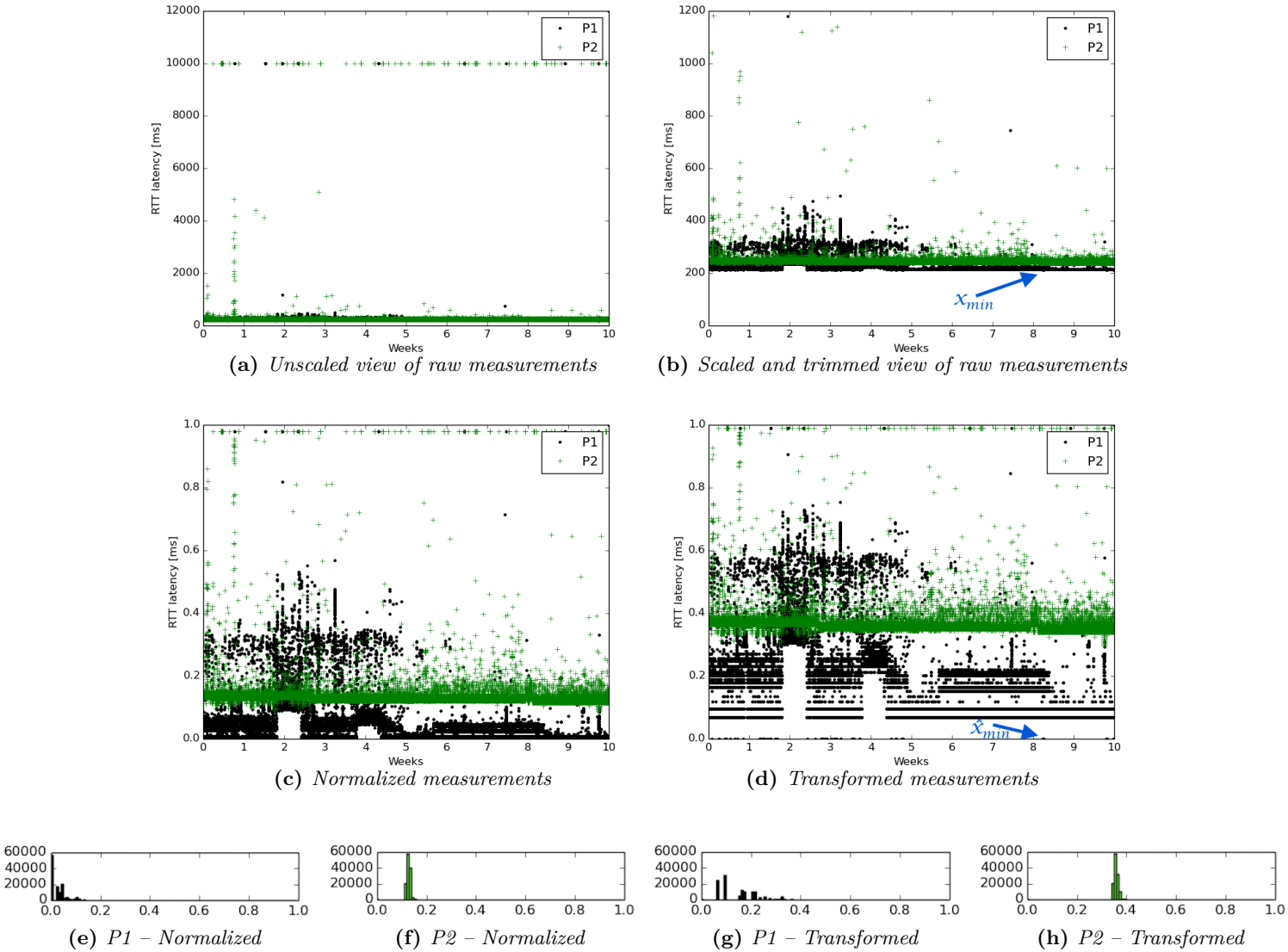
$$x_{\mathrm{min}}^{l,v,f,b} = \min_{n,p}\{x_n^{l,v,f,b,p}\},\ n \in [1, 2, \ldots, N]\,, p \in P \tag{6.4}$$

This normalization yields highly–clustered measurements with different positions of central points across CSPs, locations and even protocol layers (Figure 6.7c). However, for the purpose of subsequent benchmarking calculations, the values need to be spread out such that the large latency values represent one end of the interval and small values the other end.

The *square root transformation* has the desired properties [101], as shown using transformation outcome in Figure 6.7d. This transformation changes the distribution of measurements, but, when applied consistently, does not affect validity of the methodology. The entire formula, selected for measurement preprocessing, is as follows:

$$\hat{x}_n^{l,v,f,b,p} = \sqrt{1 - \frac{x_{\mathrm{min}}^{l,v,f,b}}{x_n^{l,v,f,b,p}}},\ n \in [1, 2, \ldots, N] \tag{6.5}$$

Measurements are now spread out across the standard $[0, 1]$ range. Subtraction from 1 is to retain the logical order, where the minimum becomes 0 and outliers appear at the other end of the range. The latency values are now comparable across locations.



(a) *Unscaled view of raw measurements* **(b)** *Scaled and trimmed view of raw measurements*

(c) *Normalized measurements* **(d)** *Transformed measurements*

(e) *P1 – Normalized* **(f)** *P2 – Normalized* **(g)** *P1 – Transformed* **(h)** *P2 – Transformed*

Figure 6.7: ***Measurement preprocessing for Latency–based benchmarking.*** *Values get normalized using minimum recorded value x_{min} and transformed using square root, as shown in respective steps (a–d) and resulting histograms (e–h). This example shows RTTs of SQL request–response interactions between Dublin DC frontend server and Tokyo DC backend server of both CSPs (P1 and P2), as recorded over 10 weeks.*

6.3.3 Benchmarking Procedure

The goal of benchmarking is to report how well different systems perform under the given constraints. In practice, benchmarks are used to guide decisions about the most economical provisioning strategy or to gain insights into performance bottlenecks [63]. The specific usage of benchmarking is dictated by concrete use cases such as financial trading or instant messaging. Whereas the former mandates very low latency with no tails, the latter allows for arbitrary high and volatile latency below a certain physiological threshold. Such requirements can be expressed via metrics, which are used throughout our benchmarking process (summarized in Figure 6.8).

Using notation from Table 6.6, measurements for benchmarking are obtained between measurement sources $s \in S$ and a destinations $d \in D$. s and d can both be any of the $v \in V, f \in F$ or $b \in B$ (i.e., benchmarking works both upstream and downstream). However, not all combinations are valid, as VPs cannot directly access backend resources. This Section considers only upstream benchmarking (i.e., d is a CSP resource f or b).

Selection of metrics

Given the sources, destinations and transformed measurements with desired characteristics, we can calculate the metric value. Metric $m \in M$ expresses a latency–related application requirements such as low or stable latency. Arbitrary set of metrics M can

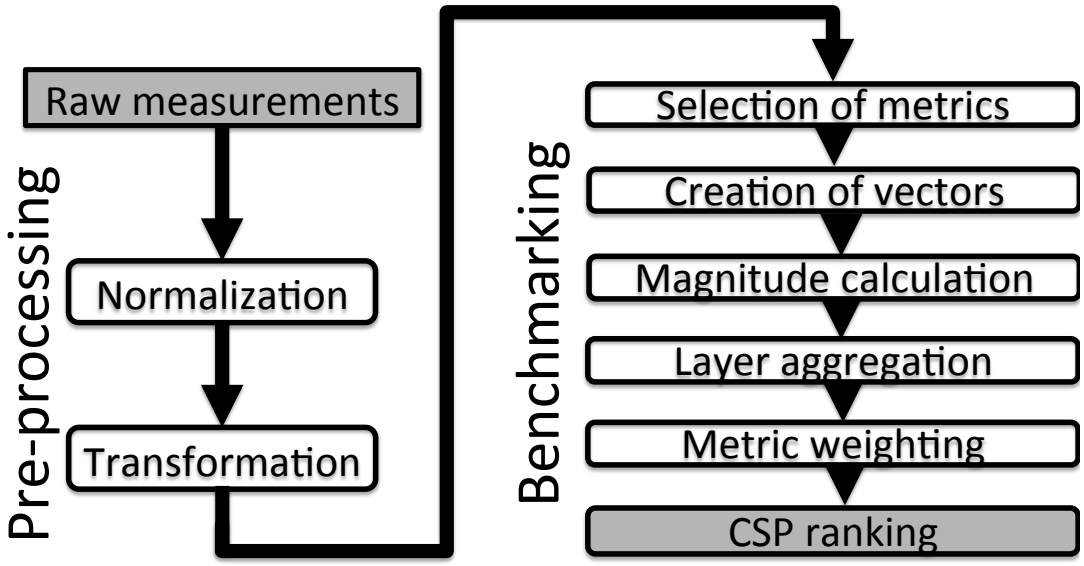


Figure 6.8: ***Procedure to transform raw measurements into CSP rank.***

be composed, to reflect composite application requirements well. In this Section, we use descriptive–statistics metrics [70]. The following two examples show *mean* and *standard deviation* metric, used to express requirements related to latency average and variability:

$$\overline{X}^{l,v,f,b,p} = \frac{1}{N} \sum_{n=1}^{N} \hat{x}_n^{l,v,f,b,p} \tag{6.6}$$

$$X_{std}^{l,v,f,b,p} = \sqrt{\frac{1}{N-1} \sum_{n=1}^{N} (\hat{x}_n^{l,v,f,b,p} - \overline{X}^{l,v,f,b,p})^2} \tag{6.7}$$

Metric vectors

Given $|M|$ metrics of interest, we create a set of $|M| \cdot |P| \cdot |L|$ vectors $\vec{v}$ in a $|S|$–dimensional space, where $|S|$ is a number of measurement sources s (Figure 6.9 gives an example).

Vector components are metric values, calculated between source location s_i and CSP p's destination d using protocol l:

$$\vec{v}_{\overline{X}}^{l,d,p} = (\overline{X}^{l,d,p,s_1}, \overline{X}^{l,d,p,s_2}, \ldots, \overline{X}^{l,d,p,s_{|S|}}) \tag{6.8}$$

Here we calculated a vector consisting of $|S|$ means of latency at protocol layer l between $|S|$ sources and CSP p's destination d.

A magnitude of such vector, calculated as *Euclidean norm*, summarizes the performance under the selected metric over all measurement source locations S:

$$\|\vec{v}_{\overline{X}}^{l,d,p}\| = \sqrt{(\overline{X}^{l,d,p,s_1})^2 + (\overline{X}^{l,d,p,s_2})^2 + \cdots + (\overline{X}^{l,d,p,s_{|S|}})^2} \tag{6.9}$$

The metrics, under which CSP performs well, have a vector magnitude close to zero. This also occurs in the case of co–located resources, where latency often approaches 0 ms (e.g., frontend and backend in the same DC). Thus, co–located deployment usually dominates the comparison, which is desirable.

Note that an outstanding source does not influence the resulting CSP ranking, as it hurts all CSPs to the same extent. Thus, there is no need to limit the set of considered sources based on the actual geographic relevance.

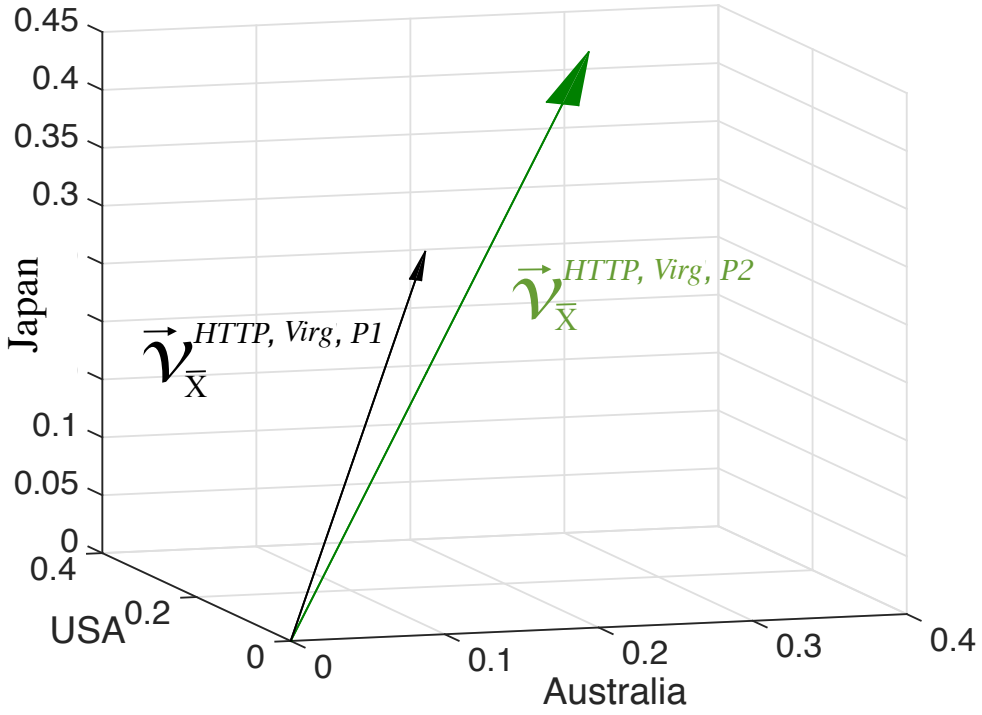


Figure 6.9: ***Example metric vectors of CSPs P1 and P2 in a 3–dimensional space.*** *Means of HTTP latency measurements of similarly–located Virginia DCs from 3 VPs (Australia, USA, Japan) were used. The magnitude of P1's vector is smaller and, as per our methodology, benchmarking favors P1 in the subsequent comparisons.*

Protocol layer aggregation

As the benchmarked application is usually built on top of the standard network protocol stack, different protocol layers take turns in issuing round trips between application endpoints. As such, every involved protocol $l \in L$ needs to perform well (Figure 2.8). To reflect this, we aggregate magnitudes of the per–protocol–layer vectors using the following multiplication, which ensures that all involved protocols perform well:

$$\|\vec{v}_m^{d,p}\| = \prod_{l \in L} \|\vec{v}_m^{l,d,p}\| \tag{6.10}$$

This approach can be extended to prioritize or penalize some layers, which certain use cases may require.

Metric weighting

Depending on tenant's and application's needs, some metrics may be of a higher priority than others. Weight $w_m \in [1, \ldots, MAX]$ is proportional to the m–th metric's influence on application performance, i.e., weight 1 is assigned to a metric that stands

for application requirement, which, if not satisfied, does not impact performance significantly; and weight MAX is assigned to a metric having a strong impact. Given $|M|$ metrics of interest, the $|M|$ weights $w_m \in [1, \ldots, MAX], m \in [1, 2, \ldots, |M|]$ are created and normalized as:

$$\widehat{w}_m = \frac{w_m}{\sum_{m=1}^{|M|} w_i} \tag{6.11}$$

CSP ranking

Finally, CSPs are scored by a following weighted sum of vector magnitudes:

$$r^{d,p} = \widehat{w}_1 \|\vec{v}_{m1}^{d,p}\| + \widehat{w}_2 \|\vec{v}_{m2}^{d,p}\| + \cdots + \widehat{w}_{|M|} \|\vec{v}_{m|M|}^{d,p}\| \tag{6.12}$$

The recommended CSPs are then the ones with low–ranked scores:

$$r^{d,p_1} \leq r^{d,p_2} \leq \cdots \leq r^{d,p_p} \tag{6.13}$$

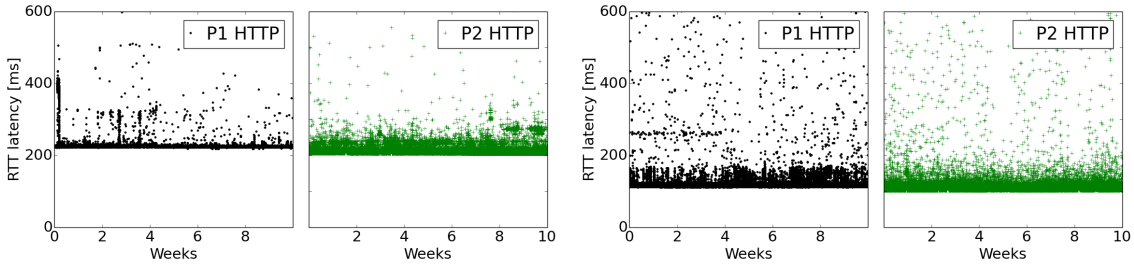
6.3.4 Performance Evaluation

This Subsection discusses an example case study of a hypothetical tenant that wants to migrate an application to the public Cloud environment. The application is a latency–sensitive TCP/HTTP web container serving only static webpages (previously used in [86]). The tenant considers Virginia locations of the public CSPs P1 and P2 for deployment. The nature of the application requires low–to–moderate latency, preferably stable, in a sense of the following weights: $median = 5 = MAX$, $standard\ deviation = 1$, $coefficient\ of\ variation = 1$.

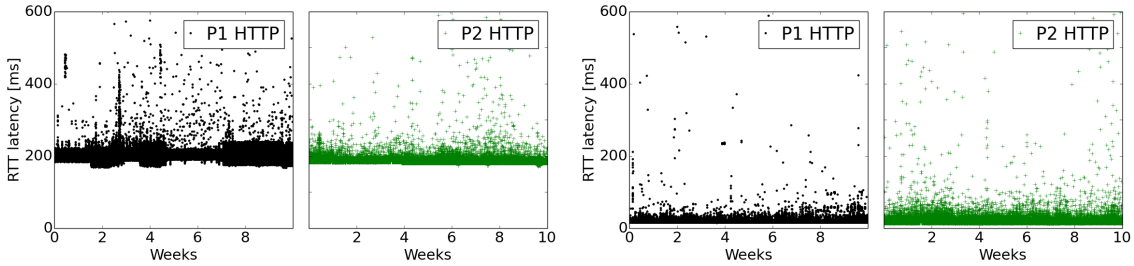
Figure 6.10 shows the input HTTP measurements and Figure 6.11 the input TCP measurements. After normalization and transformation, benchmarking proceeds according to the steps described in Figure 6.8. Table 6.7 calculates magnitudes of the metric vectors. P2 has over 30% lower median magnitude for both involved protocols,

Table 6.7: ***Calculated metric–vector magnitudes*** $\|\vec{v}\|$.

CSP	protocol	med	std	CV
P1	HTTP	0.771	0.123	0.303
	TCP	0.998	0.093	0.231
P2	HTTP	0.518	0.195	0.773
	TCP	0.727	0.131	0.836

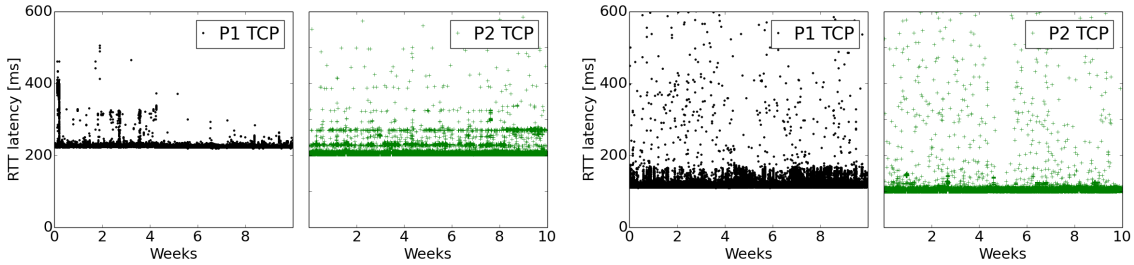


(a) *HTTP (AustraliaVP)* **(b)** *HTTP (Czech VP)*

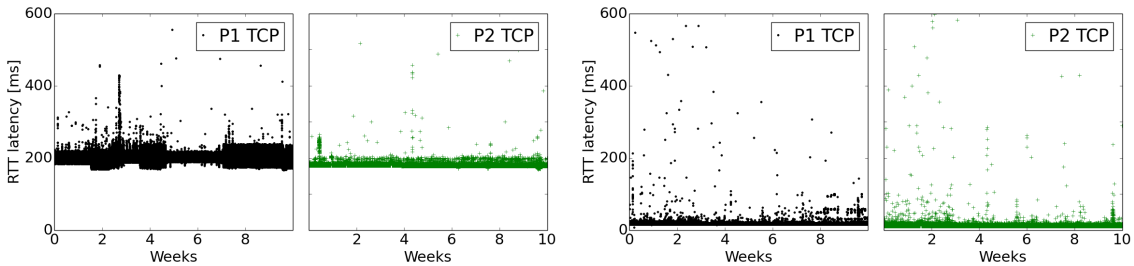


(c) *HTTP (Japan VP)* **(d)** *HTTP (US VP)*

Figure 6.10: ***Case study input data – HTTP RTTs to Virginia DC as observed by four VPs.*** *CSP P2's DC displays a lower median latency and CSP P1's DC a more–stable latency, according to observations of the majority of VPs. These observations are quantified and aggregated during the stages of benchmarking process.*



(a) *TCP (AustraliaVP)* **(b)** *TCP (Czech VP)*



(c) *TCP (Japan VP)* **(d)** *TCP (US VP)*

Figure 6.11: ***Case study input data – TCP RTTs to Virginia DC as observed by four VPs.*** *CSP P2's DC displays a lower median latency and CSP P1's DC a more–stable latency, according to observations of the majority of VPs. These observations are quantified and aggregated during the stages of benchmarking process.*

Table 6.8: ***Calculated CSP score.*** *P2 is the recommended CSP.*

CSP	r^{Virginia}
P1	0.56
P2	0.36

indicating that it provides lower latency across the communication protocol stack. In the case of both standard deviation and the coefficient of variance, P1 scored notably better and, as such, is deemed to provide more stable latency. We can visually compare the corresponding vectors in a 4–dimensional vector space, analogous to Figure 6.9.

Next, we aggregate both layers via multiplication and calculate the weighted sum using the following weights:

$$\widehat{w}_{med} = \frac{5}{7}, \;\; \widehat{w}_{std} = \frac{1}{7}, \;\; \widehat{w}_{CV} = \frac{1}{7} \tag{6.14}$$

The weights reflect the primary need for low latency and the secondary need for stable latency. Weights are plugged into the weighted sum formula:

$$r^{\text{Virginia},p} = \widehat{w}_{med}\|\vec{v}_{\text{med}}^{\text{Virginia},p}\| + \widehat{w}_{std}\|\vec{v}_{\text{std}}^{\text{Virginia},p}\| + \widehat{w}_{CV}\|\vec{v}_{\text{CV}}^{\text{Virginia},p}\| \tag{6.15}$$

Table 6.8 shows that CSP P2 is recommended for hosting the latency-sensitive web application in Virginia. Importantly, CSP P2 had a sufficiently lower latency and thus scored better overall, despite the more stable–latency at CSP P1. In Figures 6.10 and 6.11, a lower median latency at P2 can be observed through tens of milliseconds difference at all VPs. Higher latency–stability at P1 is caused mainly by Australia and Czech VP's observations at both layers. In contrast, Japan VP experienced a higher stability at P2. US VP's observations were least significant, owing to a physical proximity of this VP to the DCs considered.

Weights are important for Latency–based benchmarking. In the case of Virginia, the measurements alone yield the opposite result – by setting all weights to 1, P1's score becomes 0.85 and P2's score becomes 1.05. Requirements–agnostic benchmarking would thus place the application to the environment suboptimal for its operation. This demonstrates the importance of considering application needs when benchmarking CSPs.

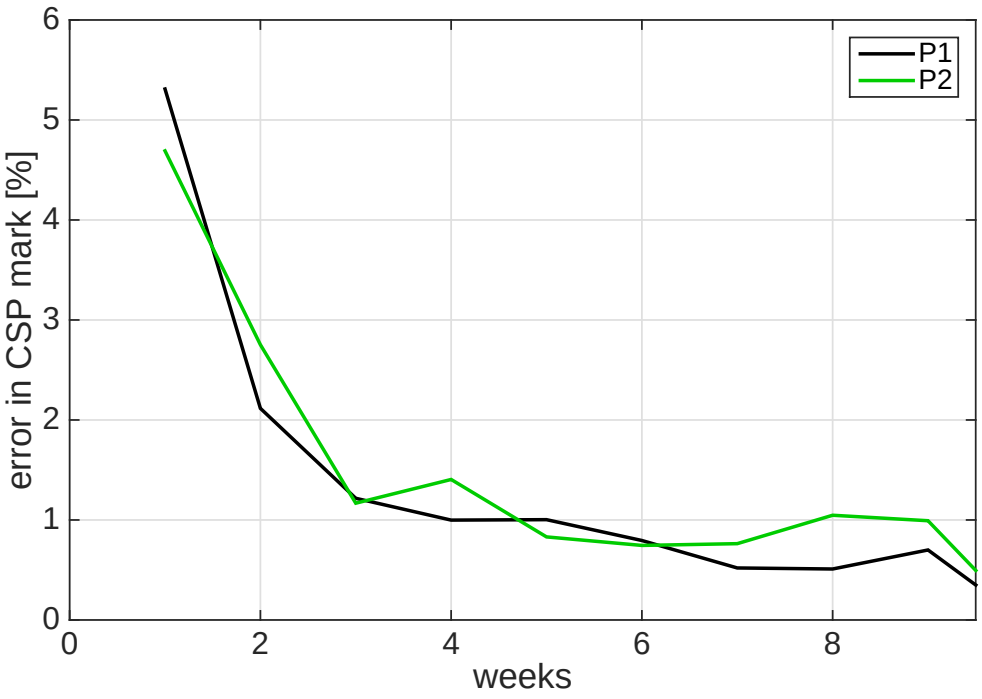


Figure 6.12: ***Error function of benchmarked CSP's score with growing dataset.*** *Our case study indicates that at least 3–6 weeks of measurements are recommended for the accurate Latency–based benchmarking.*

6.3.5 Amount of measurements

The amount of measurements needed for accurate CSP benchmarking depends on the particular CSP's stability. Figure 6.12 shows changing error in CSP scores with growing amount of the measurements. The error was $\leq$ 4.1% at 3 weeks and $\leq$ 2.2% at 6 weeks of measurements. In general, stabilization may not arise even with a greater amount of measurements – due to changing DC and network conditions. That said, our long–term measurements of major public CSPs indicate that an amount of measurements smaller than 3 weeks is insufficient due to week–of–month and day–of–week distortions.

6.3.6 Measurements summary

As an overview of the major public CSP–latency behavior, Tables 6.9 and 6.10 show a summary of computed magnitudes of various metric vectors for the entire CLAudit prototype deployment (Subsection 5.3.3). Table 6.9 shows the biggest differences in both mean and median frontend latency between the CSPs are at the US DCs. The highest latency variance and biggest deviations were observed at Singapore DC, again with a big relative difference between the CSPs (over 50% in the case of TCP latency variance). P2 has also an excess variance of HTTP latency at Virginia DC.

Table 6.9: ***Summary of vector magnitudes for HTTP–based frontends.*** *Included are Mean, Median, Variance and Standard deviation.*

			Frontend DC location							
			California		Dublin		Singapore		Virginia	
			P1	P2	P1	P2	P1	P2	P1	P2
Metric	Mean	HTTP	0.823	0.556	0.544	0.684	0.689	0.634	0.780	0.562
		TCP	0.853	0.514	0.545	0.590	0.718	0.596	0.998	0.736
	Median	HTTP	0.809	0.531	0.511	0.668	0.693	0.604	0.771	0.518
		TCP	0.842	0.491	0.564	0.585	0.723	0.571	0.998	0.727
	Variance	HTTP	0.009	0.014	0.013	0.008	0.023	0.031	0.008	0.024
		TCP	0.009	0.013	0.014	0.008	0.016	0.033	0.005	0.011
	Deviation	HTTP	0.127	0.155	0.148	0.125	0.201	0.230	0.123	0.195
		TCP	0.126	0.143	0.152	0.124	0.171	0.239	0.093	0.131

Table 6.10: ***Summary of vector magnitudes for SQL–based backends.*** *Included are Mean, Median, Variance and Standard deviation.*

			Backend DC location							
			California		Dublin		Singapore		Tokyo	
			P1	P2	P1	P2	P1	P2	P1	P2
Metric	Mean	SQL	1.130	1.015	1.106	1.272	0.422	0.677	0.447	0.513
		TCP	0.653	0.768	1.114	1.301	0.858	0.986	0.488	0.640
	Median	SQL	1.136	1.018	1.107	1.258	0.347	0.680	0.429	0.534
		TCP	0.551	0.997	1.114	1.299	0.797	1.149	0.450	0.636
	Variance	SQL	0.014	0.012	0.033	0.003	0.039	0.049	0.011	0.023
		TCP	0.153	0.169	0.035	0.007	0.026	0.154	0.013	0.020
	Deviation	SQL	0.140	0.140	0.233	0.066	0.252	0.242	0.136	0.167
		TCP	0.414	0.447	0.245	0.097	0.205	0.408	0.153	0.174

Backend results (Table 6.10) are heavily influenced by the DC and inter–DC network design. Big differences in latency deviation and variance between the CSPs were observed at Dublin DC (TCP and SQL latency) and Singapore DC (TCP latency). There was also a big disagreement between SQL and TCP average latency and also the latency variability at California DCs of both CSPs.

The magnitudes, listed in the two Tables, allow to assign arbitrary weights to metrics and observe changing results of benchmarking. Or to benchmark an application with multiple frontends or backends distributed across DCs and locate clients that such deployment serves best.

6.3.7 Conclusion

In this Section, we presented a methodology for, and a practical example of, Cloud–service benchmarking using multidimensional latency measurements. No similar longitudinal latency–based benchmarking methodologies exist at the moment, unfortunately preventing a comparison.

To make comparisons of future benchmarking methodologies easier, additional normalizations in later stages of the benchmarking process can confine the vectors and comparisons to the n–dimensional cube of a unit size.

This practical study clearly shows that selecting the best fitting CSP is not straightforward and detailed application QoS requirements must be known a priori to obtain a clear picture, as different Cloud services outperform others in different aspects or deployment locations.

The possible applicability of such benchmarking methodology is widespread: selecting a CSP, monitoring its performance, determining a workload split among CSPs including real–time adjustments, or serving as input to dynamic auctioning or pricing of Cloud services.

Clearly, an improvement of the methodology is possible – for example by integrating other measurable parameters such as throughput, computation or storage performance, or by evaluating more sophisticated applications. Furthermore, despite the practical application demonstrated, a more elaborate test, considering price differences and evaluating the benefit of having selected a particular deployment based on a benchmarking recommendation, would be needed to fully confirm applicability of this methodology.

6.4 Cloud Connectivity Optimization

6.4.1 Introduction

In this Section, we present a *Cloud–connectivity optimization scheme*, formalized as a *binary Mixed–Integer Linear Program (MILP)*. The scheme, to be deployed on gateway middleware, leverages available network information and models Cloud status through statistical measures of latency, throughput and traffic costs. These then guide dynamic Cloud–bound traffic assignments. In turn, adverse network effects on packets traversing a home boundary are minimized by leveraging up–to–date information about performance variations of candidate DC connections. Our scheme exhibits the following benefits:

- mitigates impact of Internet–path and Cloud outages;
- diverts traffic from degraded DCs;
- reduces adverse network effects on existing connections;
- precalculates assignment of future Cloud connections.

We demonstrate that such optimization could bring significant benefits in terms of quality and reliability of the Cloud service to individual users. We add to the state–of–the–art a solution that operates at the network edge, uses readily–available data, is preemptive, avoids vendor lock–in, allows for triggered or periodic execution and operates at a sub–request granularity while considering both upstream and downstream path performance. We evaluate the scheme merits by 70–day simulation – using two common scenarios and a typical application mix, hosted in five representative locations worldwide and serviced by eight globally–deployed DCs of two major public CSPs – Amazon and Microsoft (from now on referred to as using randomized order and obfuscated names – P1 and P2). Our small–scale evaluation using smart homes confirms that adverse network effects are reduced and outages, as well as poorly–performing DCs on the Cloud end are avoided – for the benefit of smart–home users, Cloud tenants and CSPs. To take full advantage of the scheme in larger deployments, practical implementation concerns have to be addressed and a good latency–prediction model has to be used. Our main contributions are:

- A proposed traffic–assignment optimization scheme;
- An experimental evaluation over a large open dataset;
- A discussion of practical implementation.

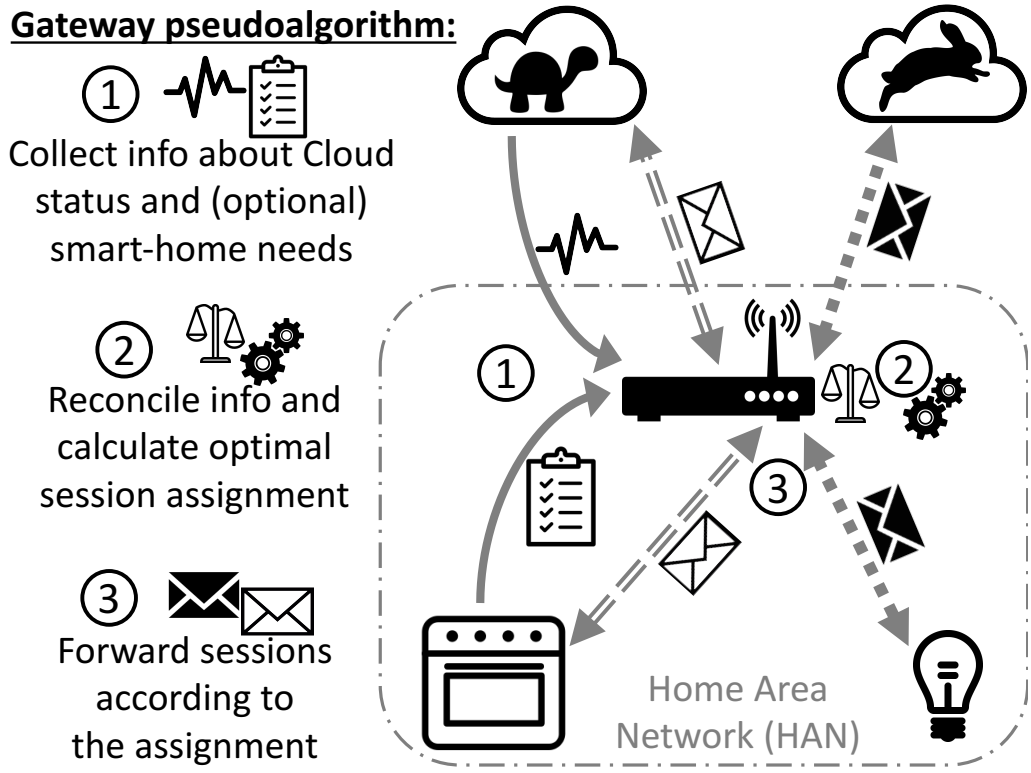


Figure 6.13: ***Smart–home system outline.*** *Smart–home gateway observes the status of candidate Cloud DCs and, optionally, the needs of smart–home devices (both denoted by solid arrows). This information is then used for mitigating impact of Cloud outages and degradations, as well as for periodic or triggered assignment–optimization of existing and future smart–home sessions interacting with Cloud services (denoted by double–dashed and square–dot arrows).*

6.4.2 Model and Optimization

The system of interest, outlined in Figure 6.13, consists of smart–home devices that run applications, which interact with Cloud services over the network via a smart–home gateway. Smart–home applications can be classified according to relationship to adverse network conditions, i.e., applications can be latency–sensitive (realtime control), jitter–sensitive (gaming), loss–sensitive (video on demand) or throughput–intensive (streaming). To capture this diversity, we model the needs of a set of smart–home application sessions I that interact with Cloud services. Sessions I are described by data rates R, average packet sizes P (in both upload and download directions M) and by sensitivity W to adverse network conditions. We also consider a monetary budget b to control Cloud traffic costs. Table 6.11 summarizes the notation.

We then describe a set of candidate Cloud DCs J using their properties, as visible to the smart home. DCs J are described by download and upload throughputs T; cost of download traffic c and $|K| = 3$ representative measures of adverse latency conditions:

Table 6.11: ***Summary of optimization notation.***

Term	Definition
I	Set of smart–home sessions
J	Set of candidate DCs
N	Length of RTT timeseries
K	Set of adverse network effects: τ, μ and σ
M	Set of traffic directions: upload and download
$x_{j,n}$	RTT measurement of j–th DC at n–th time instant
$r_{i,m}$	Data rate of session i in direction m
$p_{i,m}$	Average packet size of session i in direction m
$w_{i,k}$	Sensitivity of i–th session to k–th adverse effect
b	Remaining financial budget for Cloud traffic
c_j	Downstream traffic costs at j–th DC
$t_{j,m}$	Throughput of j–th DC in direction m
$a_{j,k}$	Size of k–th adverse network effect at j–th DC
$s_{i,j}$	Traffic fraction of i–th session, assigned to j–th DC
$s'_{i,j}$	Previous traffic fraction of i–th session to j–th DC
f	Objective function – cumulative adverse network effect

timeout rate τ, latency mean μ and standard deviation σ (see examples in Figure 6.14). These three can be conveniently derived from the j–th DC's RTT timeseries $\vec{x}_j$ of length N, obtained by probing conducted by smart–home gateway itself or pulled from a 3$^{\text{rd}}$–party service. μ and σ are calculated using timeseries normalized and transformed across all DCs (Equation 6.16). That is to unify range and to reduce bias induced by varying distances between smart homes and candidate DCs.

$$\hat{x}_{j,n} = \sqrt{1 - \frac{\min\{x_{j,n}\}}{x_{j,n}}}, \;\; n = 1 \dots N, \;\; \forall j \in J \tag{6.16}$$

τ is transformed using a square root operation, as it tends to be highly clustered (Equation 6.17). For convenience, τ, μ and σ are organized as columns of matrix A (Equation 6.17, 6.18 and 6.19).

$$\tau_j = a_{j,\tau} = \sqrt{\frac{\#\text{timeouts}(\vec{x}_j)}{N}}, \;\; \forall j \in J \tag{6.17}$$

$$\mu_j = a_{j,\mu} = \frac{1}{N} \sum_{\forall n} \hat{x}_{j,n}, \;\; \forall j \in J \tag{6.18}$$

$$\sigma_j = a_{j,\sigma} = \sqrt{\frac{1}{N-1} \sum_{\forall n} |\hat{x}_{j,n} - \mu_j|^2}, \;\; \forall j \in J \tag{6.19}$$

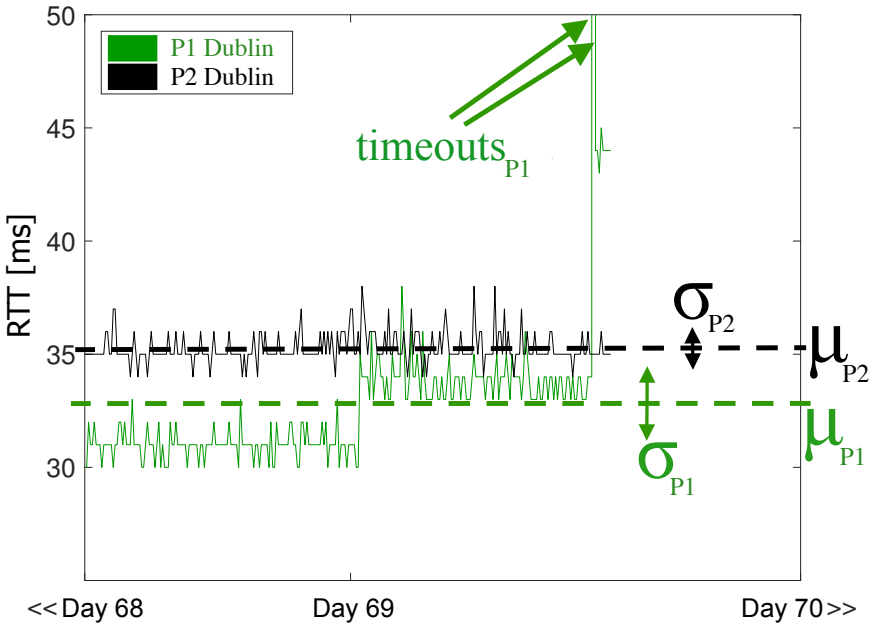


Figure 6.14: ***Example of adverse network effects.*** *Derived from j–th DC's RTT timeseries $\vec{x}_j$ are timeout rate τ_j, latency mean μ_j and standard deviation σ_j that represent the ever–present adverse network effects whose optimal combination across a set of candidate DCs J is sought.*

Given the smart–home and Cloud models, we formulate the optimization task as a binary MILP (Objective function 6.20). The goal of the Objective function f is to minimize cumulative adverse network effect (i.e., amount of latency, deviation and timeouts) on packets traversing a home network boundary. The optimized variable $s_{i,j}$ represents session i's traffic fraction, assigned to DC j. Constraints ensure that only DCs capable of serving respective sessions are considered (Constraint 6.23), selected sessions do not undergo optimization (i.e., are not steered or split – Constraints 6.24 and 6.25), DC throughputs are sufficient (Constraints 6.26 and 6.27) and remaining budget is sufficient (Constraint 6.28). The said constraints can be relaxed depending on information availability. Note that the Objective function consists of only RTT–derived metrics and is unitless, since it contains one ratio and two normalized quantities. DC–upstream and downstream throughputs and traffic costs are modeled as constraints – as such, they restrict the solution space and thus affect the optimal DC selection. We refrained from adding these to the Objective function due to their incomparable units ($, kbps).

In the worst case, presence of Constraints 6.25 changes the problem complexity from $\mathcal{P}$ to $\mathcal{NP}$–hard. In that case, a practical program computation is thinkable only for up to our considered deployment scales, e.g., a network with a moderate number of Cloud–bound sessions I and several candidate DCs J. In such case the program computation is feasible using conventional solvers.

$$\underset{s\in\mathbb{R}}{\operatorname{argmin}} f(s) = \underset{s\in\mathbb{R}}{\operatorname{argmin}} \sum_{\forall i}\sum_{\forall j}\sum_{\forall k}\sum_{\forall m} w_{i,k}a_{j,k}\frac{r_{i,m}s_{i,j}}{p_{i,m}} \tag{6.20}$$

subject to

$$s_{i,j} \geq 0,\ \ \forall i \in I,\ \ \forall j \in J \tag{6.21}$$

$$\sum_{\forall j} s_{i,j} = 1,\ \ \forall i \in I \tag{6.22}$$

$$s_{i,j} = 0,\ \ \forall (i,j) : j \text{ incapable of serving } i \tag{6.23}$$

$$s_{i,j} = s'_{i,j},\ \ \forall i : i \text{ cannot be steered},\ \ \forall j \in J \tag{6.24}$$

$$s_{i,j} \in \{0,1\},\ \ \forall i : i \text{ cannot be split},\ \ \forall j \in J \tag{6.25}$$

$$\sum_{\forall i} s_{i,j}r_{i,m} \leq t_{j,m},\ \ \forall j \in J,\ \ m = \text{download} \tag{6.26}$$

$$\sum_{\forall i} s_{i,j}r_{i,m} \leq t_{j,m},\ \ \forall j \in J,\ \ m = \text{upload} \tag{6.27}$$

$$\sum_{\forall i}\sum_{\forall j} s_{i,j}r_{i,m}c_j \leq b,\ \ m = \text{download} \tag{6.28}$$

The program computation is preemptive in that it reassigns sessions capable of being reassigned throughout their duration. It is either periodic (according to a configured timer) or triggered by a significant change in Cloud status or smart–home needs. The entire optimization scheme is outlined in Figure 6.15. An example of the optimization–scheme behavior around a period of DC degradation is depicted in Figure 6.16.

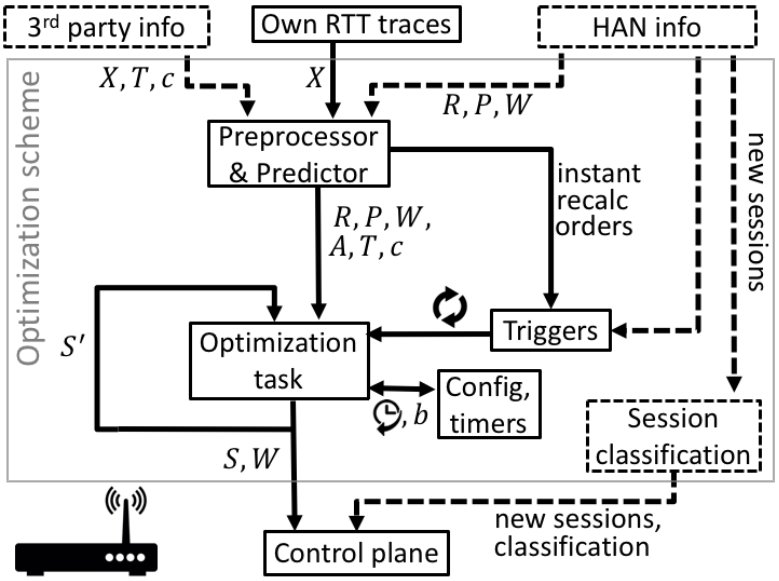


Figure 6.15: ***Information flow within a smart–home gateway.*** *Cloud status information comes from gateway's own active probing or optional 3rd–party sources. An optional information about smart–home session needs comes from monitoring the home area network (HAN) activity. Optimization–task's execution can be either periodic or triggered by a significant change in HAN or Cloud status. New HAN sessions can be classified and, in turn, suitably pre–assigned to serving DCs. The optional modules and flows, unnecessary for the general idea to work, are denoted using dashed boxes and arrows.*

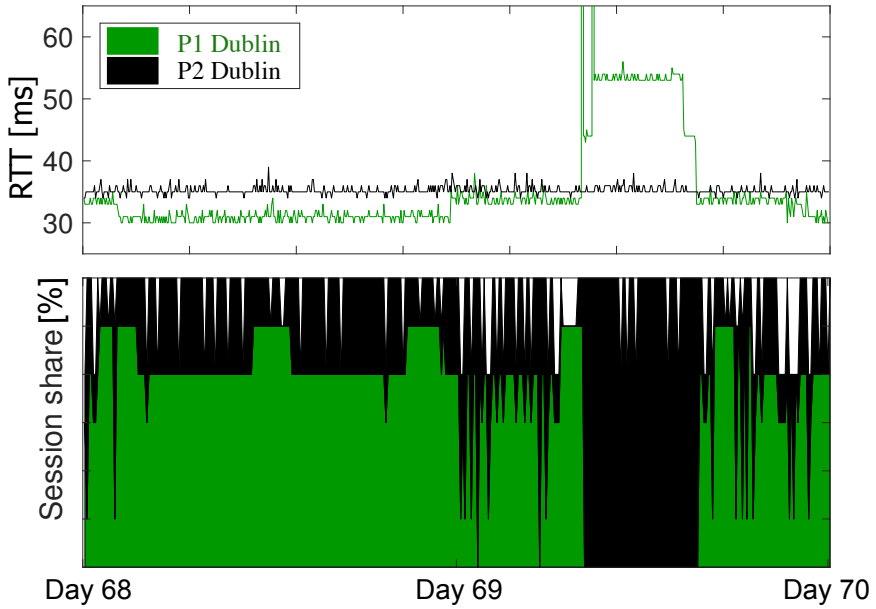


Figure 6.16: ***Optimization–scheme behavior.*** *The scheme observes preferred–DC's degrading status and, after a short delay, reacts by offloading sessions to a momentarily best–performing and suitable DC. Sessions are assigned back after the preferred–DC's status is restored. On Day 69, all six sessions were taken away of P1's DC for the period of about seven hours of excess RTTs.*

6.4.3 Performance Evaluation

We have used Dataset4 (Table 6.1) for evaluation of our proposed optimization scheme, since, VP sites therein correspond well with the current global IoT deployment (Figure 6.17). We thus emulate the smart–home perspective using PlanetLab hosts. Cloud–service instances are emulated using the actual frontend–server deployment inside four common DC locations of P2 and P1. Every smart home periodically measures TCP RTT to the eight candidate DC service instances. Recent parts of resulting timeseries $\vec{x}_j$ are periodically preprocessed (normalized, transformed and summarized, according to Equations 6.16–6.19). We have experimented with various simple prediction models that consider various amounts of history, such as *low–pass filter*, but the most recent parts of the timeseries turned out to be the best predictor (likely due to irregularity of the Internet and Cloud latency distributions). Period $N = 8$ minutes minimized the prediction error over Dataset4.

The predicted status of candidate DCs is fed to the optimization task (Function 6.20) that evaluates it against smart–home session needs and periodically (here every 8 minutes) suggests optimal session assignments.

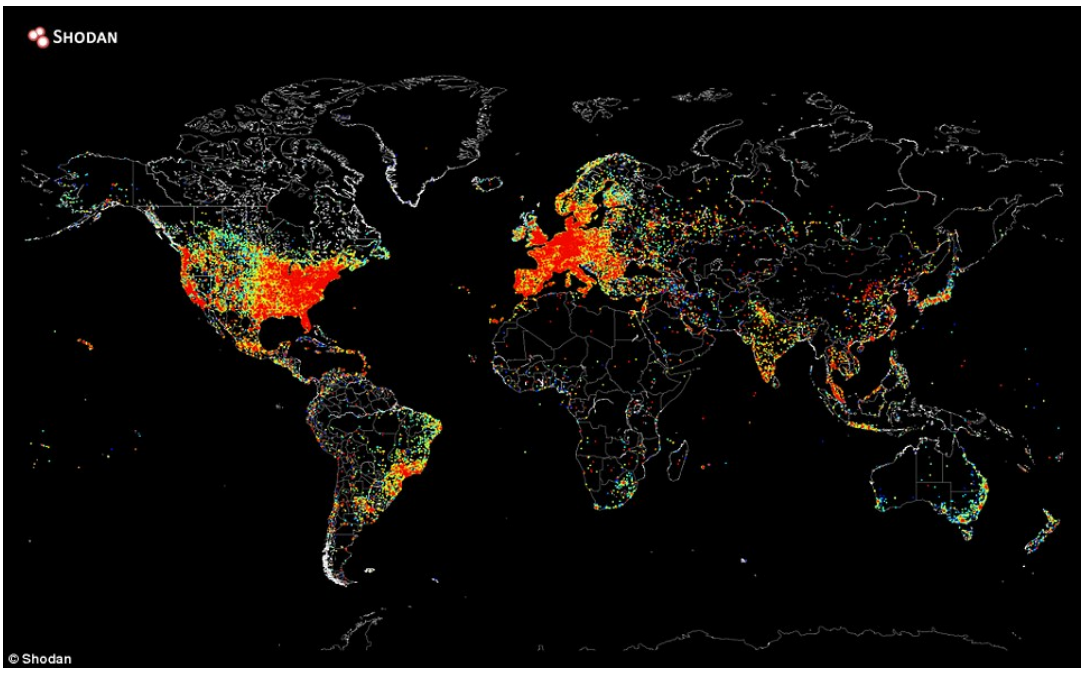


Figure 6.17: ***IoT heatmap*** *[31]. Concentration of smart devices is largest in US and Europe. Note that some countries provide limited visibility and thus skew accuracy of this Figure.*

The simulated smart–home application mix (described in Table 6.12) consists of six applications: web camera, VoIP phone, HDTV, over–the–air (OTA) device update, large file download and interactive home–control application. Every application has a representative usage pattern that ignores daylight savings and weekends. This usage pattern is neither known to, nor predicted by our optimization scheme. The application mix represents various sensitivities to timeout rate, latency and latency deviations, which are reflected in the custom–sensitivities simulation scenario w'. We have also conducted a simulation scenario of default sensitivities w'', under which applications do not express any sensitivity. To discern the influence of adverse network effects, we do not constrain the solution search space – by setting negligible traffic costs $\vec{c}$, large throughputs T and large budget b. We also assume that any DC j can serve any session i.

In the rest of this Section, we quantify the optimization merits by contrasting performance of the optimization scheme with performance of an empirically–derived most preferred DC for every smart home. A preferred DC is defined as the DC receiving the largest traffic share from the smart home under the default–sensitivities w'' scenario over the 70 days (it often turns out to be one of the closest DCs to a smart home). The preferred DCs are: P2's Virginia DC for the Belo Horizonte and the Atlanta smart homes; P2's Singapore DC for the Hiroshima smart home; P1's Dublin DC for the Prague smart home and P1's Singapore DC for the Melbourne smart home.

Merits on Datacenter end

Mitigating impact of Internet–path outages, Cloud outages and degraded DCs reduces number of helpdesk calls, technician visits and SLA violations. The optimization scheme achieves so by observing failed or degraded network conditions and subsequently updating an assignment. Smart home thus avoids experiencing many outages and degradations (as shown using performance against ground truth – Table 6.13), which could otherwise render smart devices unusable or unsafe. From a perspective of CSP, gateways naturally divert traffic to a different DC, since degraded network conditions, once observed, are less prone to be selected by the optimization task.

We define u–th DC's *outage* (Formula 6.29) as a period of 100% timeout rate observed by smart home. To rule out smart–home outages as a cause, the smart home simultaneously has to see at least one other candidate DC online.

Table 6.12: ***Smart–home application mix and two latency–sensitivity simulation scenarios.***

Need	Webcam	VoIP phone	HDTV	OTA device update	Large download	Home control
$r_{i,download}$	1kbps	87kbps	4.5Mbps	216kbps	5Mbps	128kbps
$r_{i,upload}$	256kbps	87kbps	1kbps	1kbps	1kbps	128kbps
$p_{i,download}$	256B	218B	1500B	1500B	1500B	128B
$p_{i,upload}$	256B	218B	1500B	1500B	1500B	128B
Can be split	Yes	No	Yes	Yes	Yes	No
Can be steered	Yes	No	Yes	Yes	Yes	No
Usage pattern	always–on	30 minutes every other daytime hour	1 hour mornings, 5 hours evenings	30 minutes once a week	6 hours evenings	10 minutes every 4 hours
Custom sensitivities						
$w'_{i,\tau}$	1	10	1	100	1	100
$w'_{i,\mu}$	10	100	10	1	1	100
$w'_{i,\sigma}$	10	100	10	1	1	1
Default sensitivities						
$w''_{i,\tau}$	1	1	1	1	1	1
$w''_{i,\mu}$	1	1	1	1	1	1
$w''_{i,\sigma}$	1	1	1	1	1	1

$$\tau_u = 1 \ \wedge \ \exists v \in J : \tau_v < 1 \ \wedge \ u \neq v \tag{6.29}$$

We define j–th DC's *degradation* (Formula 6.30) as a period of over 25% timeout rate, abnormally high mean latency or abnormally high standard deviation of latency.

$$(\tau_j > 0.5 \ \wedge \ \tau_j < 1) \ \vee \ \mu_j > 0.5 \ \vee \ \sigma_j > 0.5 \tag{6.30}$$

Due to the normalization (Equation 6.16), the 0.5 abnormality threshold of μ and σ does not correspond to a single fixed value. But it guarantees a sufficiently high value, since, for every smart home, we evaluate only its preferred DC, which, under normal conditions, is expected to have $\mu \approx 0$ and $\sigma \approx 0$. The timeout rate $\tau = 0.5$ abnormality threshold directly corresponds to a 25% timeout rate.

During the 70 days, according to the above definitions of outage and degradation, the five smart homes observed a total of 35.3 hours of outage (mostly Belo Horizonte), of which 8.7 hours were observed on connections to smart–homes' preferred DCs. Also observed on preferred connections were 65.5 hours of degradation. The optimization–scheme behavior in the presence of outage and degradation is validated in Table 6.13, from which we can see that the scheme was doing a good job averting packets from all problematic DCs.

Table 6.13: ***Scheme validation.*** *Percentage of averted packets and ratio between decisions actually made and decisions that could have been made to avert session from a problematic DC. The upper bounds on averted packets and avert decisions were derived from the Dataset4 according to Formulas 6.29 and 6.30.*

		Outage period		Degradation period	
Problematic DC	W	Averted packets	Avert decisions	Averted packets	Avert decisions
P2 Singapore	w'	99.99%	3/6	61.32%	17/42
	w''	99.99%	3/6	61.32%	16/42
P1 Singapore	w'	–	0/0	85.3%	510/564
	w''	–	0/0	85.64%	519/564
P2 Virginia	w'	83.87%	310/378	72.16%	1455/1950
	w''	84.16%	339/378	72.62%	1589/1950
P1 Dublin	w'	99.99%	5/6	74.51%	345/390
	w''	99.99%	6/6	74.92%	354/390

Merits on Smart–home end

We use *Relative change* of *Cumulative deltas* of f to demonstrate improvement potential inside a smart home – i.e., how much can a smart home benefit from leveraging eight DCs, contrasted to using just a single most–preferred DC. Table 6.14 shows an upper bound of such improvement, i.e., an improvement by the optimization scheme that uses perfectly predicted Cloud latency timeseries $\vec{x}_j$ as an input. In Table 6.14, we also break the improvement potential down to improvements and sacrifices made in its respective τ, μ and σ components (e.g., Atlanta, compared to relying only on its preferred DC, has avoided all timeouts and its packets accumulated 8.7%–less latency and 3.2%–less latency deviation under custom–sensitivities scenario w'). The improvement potential's upper bound is calculated in the context of our scheme and our application mix. The improvement potential comes from traffic splitting and steering among momentarily well–performing DCs, contrasted with using a fixed service instance from a smart–home vendor.

Table 6.14 also shows the magnitude of optimization's potential. It is high at distant and poorly–connected smart homes (Hiroshima, Melbourne), but low in the case of close proximity of smart home to DC (Atlanta). The lower–than–expected improvement potentials at Belo Horizonte and Prague smart homes are mainly caused by frequent tradeoff decisions (also reflected through DC traffic shares in Figure 6.18). In both w' and w'' scenarios, the optimization scheme in Prague and Belo Horizonte sacrificed a small amount of mean latency μ (i.e., sometimes decided against a DC with lower latency) in order to significantly improve timeout rate τ and standard deviation σ. As a consequence, such smart homes with lower improvement potential tend to use many DCs.

Table 6.14: ***70–day improvement potential of cumulative adverse network effect.*** *Along with changes in its τ, μ and σ components. ∞ denotes full elimination.*

Smart home	W	improvement potential			
		$\forall k$	$k = \tau$	$k = \mu$	$k = \sigma$
Atlanta	w'	5.6%	∞	8.7%	3.2%
	w''	5.6%	∞	10.2%	2.1%
Hiroshima	w'	37.7%	∞	18.9%	69.6%
	w''	39.6%	∞	18.3%	75.7%
Melbourne	w'	42.1%	∞	81.8%	3.8%
	w''	36.3%	∞	74.4%	-0.5%
Belo Horizonte	w'	15.7%	71.1%	-6.7%	66.1%
	w''	18.7%	91.2%	-6.8%	63.5%
Prague	w'	11.9%	571.6%	-2.6%	26.3%
	w''	11.6%	689.2%	-3.8%	26.5%

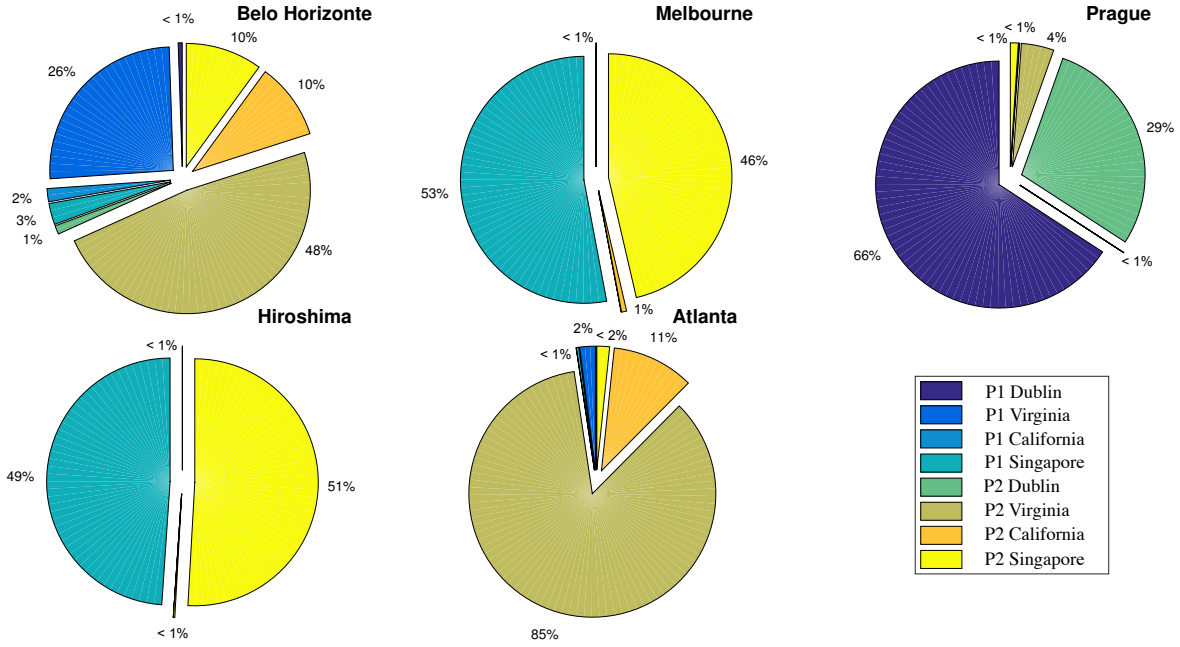


Figure 6.18: ***DC traffic shares per smart home.*** *After optimization, respective pie charts reflect 70–day cumulative shares of traffic for which the smart home has selected a particular DC under the default–sensitivities scenario w''. The optimization scheme's improvement potential is highest there where only few DC switchings occur (Melbourne, Hiroshima). Long distance, poor connectivity or need to frequently balance adverse effect tradeoffs is reflected by excess DC switching (Belo Horizonte, Prague).*

Both the custom–sensitivities w' and the default–sensitivities w'' simulation scenario (see Table 6.12) yielded similar results, suggesting low significance of sensitivities W for optimization of active (i.e., in progress) sessions. However, another use of the optimization scheme is to precalculate splits and assignments of future Cloud–bound sessions, since, as a secondary product, the optimization task yields currently–optimal assignments of active–application categories (i.e., combinations of session needs P, R and W). Once a new session requests a Cloud service, it gets classified and is then assigned according to the last known assignment corresponding to its category, as shown in Figure 6.15. This avoids suboptimal initial assignment.

6.4.4 Implementation

A convenient place for implementation of the optimization scheme (shown in Figure 6.15) is smart–home gateway, which is well–positioned to monitor and reconcile smart–home needs with the Cloud status. This Subsection describes a naïve implementation of running the optimization scheme and related modules inside the gateway's application plane. Due to the resource–intensive nature of optimal–assignment calculation

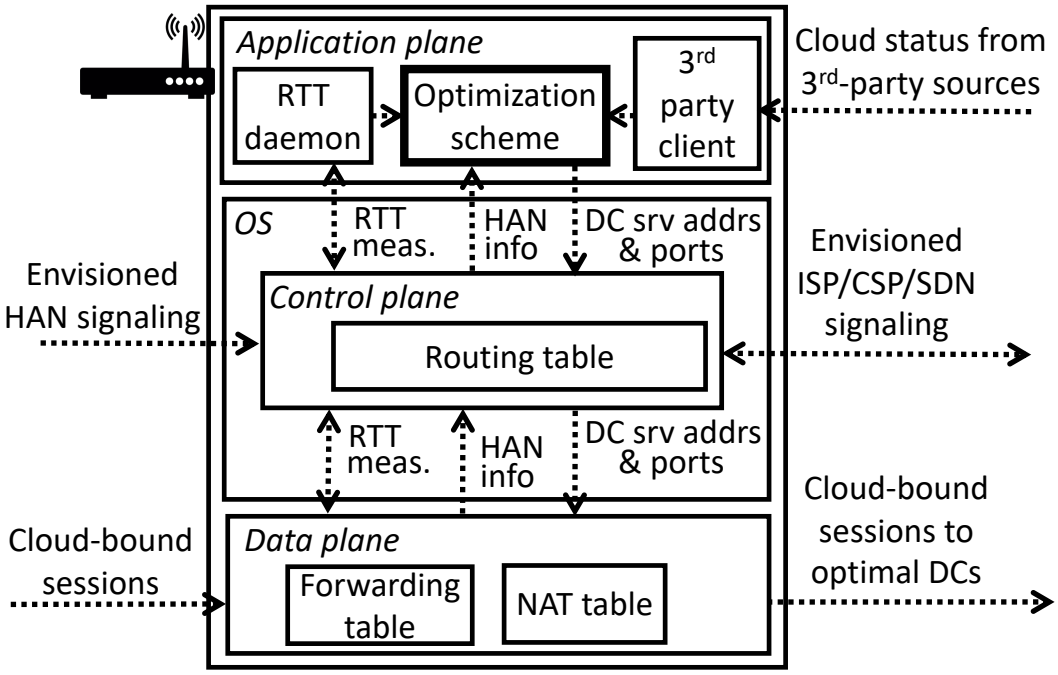


Figure 6.19: ***Full–fledged implementation using conventional router architecture.*** *Optimization scheme and related modules reside inside application plane. Modules and other sources feed their collected information into the scheme, which populates NAT table with Cloud service IP addresses and TCP/IP ports, according to which the router rewrites headers of Cloud bound traffic.*

and packet–header rewriting, a DNS–based implementation, mentioned at the end of this Subsection, can be used instead, to relieve the header–rewriting burden and preserve the End–to–end principle.

The optimization scheme can be implemented as a module inside the application plane of a conventional router architecture. So, too, can optional modules for gateway's own Cloud–RTT measurements or for Cloud–status information collection from 3rd–party sources. This full–fledged architecture is shown in Figure 6.19. The modules feed their Cloud–status information into the optimization scheme and the gateway itself monitors and profiles transit smart–home traffic for additional information for the optimization scheme. The calculated optimal–session assignments are materialized through DC service IP addresses and TCP/UDP ports, which flow into NAT table within gateway's data plane. Gateway operates in the NAT overload mode and continuously translates Cloud addresses and ports within inbound and outbound Cloud traffic (this implementation is applicable only to environments with sparse end–to–end traffic matrix). The optimization granularity is at the application session level, which can be conveniently represented using the flow 5–tuple (Source IP, Destination IP, Source Port, Destination Port, Protocol). To foster quality and interoperability of assignment decisions we envision gateway signaling

towards smart home, service CSPs and external SDN controllers.

The list of candidate DCs comes from 3rd–party sources, CSP signaling or is preconfigured. Cloud RTTs can be obtained using gateway's own probing, as latency measurements are lightweight. Cloud throughput, traffic costs and even RTTs can be obtained through CSPs dashboards [7, 3], monitors [10, 117], benchmarks [86, 124] or SLAs. Smart–home session sensitivities and capabilities of being split or steered can be conveyed using currently unused packet fields like ToS, IP options or a novel signaling protocol. These sensitivities and capabilities are to be determined by application maker or user. Average packet sizes or data rates are derived from traffic passing through the gateway. Some information can also be inferred from household usage patterns, possibly using Machine Learning. In the case when information about smart–home needs is not available, defaults are used. In the case when even Cloud status information is not available, optimization can fall back to a normal (no splitting, no steering) router forwarding. Particular sessions not to be optimized (such as CDN or cache queries) are handled by the model – using a combination of Constraints 6.24 and 6.25.

Heavy hitters or overly–sensitive sessions can trigger an optimization run and benefit from more up–to–date assignment. So, too, can the gateway itself after it observes significantly degraded Cloud status at a DC that serves an active session. In the case of an empty solution space, the last good assignment S' or fallback is used to ensure continuity.

Potentially–negative impacts of traffic steering and splitting on upstream will manifest through changing Cloud status and will thus be resolved in subsequent optimization runs. In the case of many distributed optimization–scheme instances, this can lead to undesired upstream oscillations, which must be addressed (e.g., by regulating the extent of assignment changes the optimization brings about – by using sufficiently large ratio of history to the optimization frequency). These traffic swings, resulting from en–masse deployment of the scheme, can also be resolved by regional or global external visibility and control, such as SDN signaling. Care must also be taken when dealing with session greediness or misleading 3rd–party information.

The limits of improvement, yielded by the optimization, depend on the accuracy of predicting the Cloud status and the active smart–home session set. Additional engineering enhancements to the scheme can further increase the yield, such as deferring new Cloud sessions spawned near before optimization run or using high optimization frequency to leverage fresher information.

For a commoditized service with stateful Cloud sessions capable of being split or steered, a service–state replication interval on the Cloud end has to be set such that an application function is not disrupted by an adjusted assignment. Our optimization model does not include Cloud computation and storage costs, but under the converged prices and pay–as–you–go Cloud subscription, the bill should not change as the total service usage remains the same.

To expedite optimal–assignment computations, the entire optimization calculation can be carried out elsewhere (e.g., Cloudlet or Fog) and resulting assignments delivered back to the gateway. Or just the incremental changes in smart–home needs and Cloud status can be used to calculate an incremental change to assignments (in lieu of the entire program computation). Such a change–point detection better reflects the presumably small need to adjust assignments following the initial, or sporadic, computation of the entire MILP.

An entirely different implementation option, should limited forwarding resources at the gateway become a concern, is a DNS–based implementation. In this case, optimal–assignments are realized through IP addresses in DNS responses that the smart–home gateway, as a local DNS resolver, serves to smart–home devices. These responses contain the DNS TTL information, consistent with optimization–recalculation frequency, such that devices periodically poll the gateway for new assignments. As DNS does not carry any TCP/UDP port information, the optimization would need to take place at a service level rather than session level. Also, triggered optimization would not be possible under the DNS implementation. The main advantage of this implementation is that no per–packet header rewriting of Cloud service addresses takes place, neither on the gateway nor elsewhere. The *End–to–end principle* [110] is thus not violated.

6.4.5 Conclusion

In this Section, we have introduced the assignment–optimization scheme for Cloud–bound smart–home traffic, which mitigates impact of DC outages and degradations, reduces adverse network effects on smart–home devices and precalculates assignment for future Cloud sessions.

Results of the verification and experimental evaluation using major CSPs show that the optimization scheme could have averted between 60% and 100% packets from problematic DCs. Also, the optimization's potential to reduce adverse network effects is

as much as 42%. The scheme is general enough and its applicability not limited to smart homes and Clouds. Environments with heavy network traffic and diverse application mix like campuses and small industries would perceive even greater improvements.

Smart–home users, Cloud tenants and CSPs would benefit from such a scheme. Users would obtain better services through reduced adverse network effects and CSPs would have traffic to their problematic DCs diverted naturally. Cloud tenants have long lacked feedback from CSPs regarding causes of their user issues. CSPs, on the other hand, focus on SLAs and not necessarily customer satisfaction. Our scheme globally and distributively logs Cloud status information at the network edge, allowing for accountability and proper incentives for all stakeholders. In the future, signaling among all aforementioned stakeholders might even better reconcile interests and further improve services.

Variations of the presented optimization task can be introduced by considering different measures of Cloud status, minimizing the Cloud bill as Objective function, tightly controlling the extent of changes for better stability, or approximating the optimal solution from the LP corresponding to the relaxed $\mathcal{NP}$–hard MILP. State–of–the–art solutions report performance using diverse incompatible metrics (Table 3.1), a fair comparison across these is much needed. Another challenges include development of proper Cloud–status prediction models, means of Cloud–status information sharing and instrumentation/provisioning of optimization scheme for large–scale deployments like Industry 4.0 or Smart cities.

Chapter 7

Conclusion

In this thesis, we have devised a methodology for network–based latency monitoring of Cloud Computing services, as well as three monitoring–data–driven methodologies for improving Cloud services. The data can serve various other purposes in performance engineering, data mining, network design and other fields of research.

Despite the network latency's importance for service performance and user experience, challenges such as latency unpredictability, uncontrollability and difficulties of modeling persist. Unsuccessful attempts to capture Cloud latency analytically has motivated us to devise the *large–scale multiprotocol multitarget monitoring* and employ mathematical techniques for *performance modeling and optimization*. Our approach is cost effective, has global reach, does not perturb the service and provides dependable measurements.

Using the created models, we have contributed to the following areas of Cloud Computing performance research: (1) Cloud network profiling, (2) Cloud benchmarking and (3) Cloud connectivity optimization. The results confirm the presence of many performance suboptimalities of real services and address them by leveraging extra intelligence applied on top of the multidimensional Cloud latency measurements.

7.1 Monitoring Methodology and Data Capture Platform

We have explored the state–of–the–art and design space of Cloud service monitoring techniques. We have described network latency, its Cloud traits and the problems of its proper measurement and interpretation.

We have formalized the Cloud Computing system and presented the novel *Multidimensional Cloud latency monitoring* methodology, based on active probing and record-

ing batch–RTT timeseries of multiple protocol layers of multiple Cloud service provider datacenter resources from multiple globally–dispersed redundant vantage points, both internal and external to the Cloud. As we have shown, this monitoring methodology provides increased measurement accuracy, confidence and additional insights compared to the widespread one–dimensional monitoring techniques.

Using our data capture platform *CLAudit*, the large–scale monitoring run of Amazon AWS and Microsoft Azure CSPs was conducted over five years.

7.2 Applications

We have conducted a detailed *analysis of the measurements* and identified notable Cloud failures and anomalies. The likely causes of anomalous behavior have been deduced using spatio–temporal coincidence and reasoning. We have also conducted the statistical analysis and revealed adverse network effects such as tail latency, latency variability and latency multimodality on the end–to–end Cloud paths. We have discerned the latency improvements caused by CSP investments into evolving infrastructure.

We have proposed the longitudinal *latency–based Cloud service benchmarking* methodology, capable of in–depth comparisons of Cloud Service Providers (CSPs), their datacenters and Cloud resources therein, using diverse application requirements. Comparisons take place inside n–dimensional vector spaces that characterize performance of services being compared. We designed a preprocessing method that transforms and normalizes measurements in order to reduce clustering and comparison biases. We have conducted a benchmarking case study of the real CSPs and revealed significant performance differences at granularities of Cloud service providers, datacenters and resources.

We have proposed the methodology for *optimizing Cloud connectivity* across CSPs using a gateway middleware. We have used Convex Optimization, namely binary Mixed–Integer Linear Program (MILP), to continuously minimize the value of a metric approximating cumulative adverse network effect on packets exchanged between client devices and Cloud services, all by preserving cost, throughput and session requirements. We have conducted a discrete–time simulation, using real CSP latency measurements. Results show that Cloud traffic is naturally diverted from degraded datacenters and impact of outages is mitigated. We have deduced the principal limits of the methodology and suggested its naïve implementation inside an application plane of the conventional router architecture.

7.3 Future Research Directions and Open Issues

Our monitoring architecture, data analyses, benchmarking and connectivity optimization scale well under the traditional Cloud Computing model. However, scalability enhancements and additional instrumentation are likely needed to accommodate full needs of the increasingly popular Edge/Fog Computing models with IoT–endpoint proliferation of Smart Cities or Industry 4.0.

In this thesis, we have used a fixed measurement configuration, but an increased measurement accuracy and reduced network loading can arise from employing distributed collaborative measurement architecture, using agent–based vantage points strategically issuing probes of different levels of informativeness, as was suggested in IBM's active probing scheme for problem determination in distributed systems [108].

We have considered only near–realtime measurements, but designing a way to sustainably increase measurement frequency would enable realtime measurements that many advanced applications would benefit from. Among these are Software Defined Networking (SDN) feedback–based automated applications that would enable better high–level control and preserve global network–stability objectives in face of local optimizations. Other useful applications include determining realtime–workload split among providers and dynamic auctioning of Cloud services. Fine–grained SLAs can use realtime meeasuremens as compliance data, properly incentivizing CSPs, tenants and users to act in their best interests and preserve the social welfare. Proper means of sharing such realtime Cloud latency measurements, or Cloud status in general, also remain an open question.

An increased measurement volume and extra dimensionality would allow advanced data mining and machine learning applications for Cloud–status prediction, detection of novel anomalous behaviors and classifying them as overload, security incident, seasonal swing or normal behavior. Subsequently, the SDN controller would be able to react and reconfigure the network elements to address the arisen situation accordingly.

The methodologies, presented in this thesis, can be extended by additional decision factors, e.g., financial costs, computation and storage performance, to enable multivariate optimization, benchmarking and insights. An importance of every factor remains to be determined, based on various application needs.

7.4 Impact on Industry and Academia

Our contributions both directly and indirectly lead to lower and predictable network latency, allowing for low–latency technology (e.g., Reactive AI, community Geo–spatial Cloud GIS). Even more challenging are sub–millisecond latencies for mission–critical applications (e.g., telesurgery, intelligent transportation), where the network latency has to be less or equal to the latency of body signals traversing neurological and physical pathways in the body (e.g., from brain to the eyes or from skin to nerves in the spinal column). Another uses for sub–millisecond latencies are financial trading and smart–grid control, where even a slightest time loss has a non–negligible price tag attached.

At the heart of a large socio–economic FinTech movement (e.g., distributed ledgers, cryptocurrencies, Blockchain, open banking APIs) is a distributed system needing timeliness to ensure fidelity and small attack surface, fidelity. As such, it is crucial that latencies remain low, predictable and auditable.

New large–scale infrastructures (e.g., DCs or Fog networks) can instantly obtain complex observations and insights into their real–world behavior, as well as become part of standard public Cloud benchmarks. Existing providers, tenants and users can leverage this benchmark as a data service for strict compliance monitoring, which is extremely important for continuous adaptive risk and trust of cyber–physical systems in industry automation, transportation and health technology. Home and other edge users gain additional control over their traffic destinations and application requirements, which is important given edge data volumes produced and diverse CX–oriented applications such as immersive augmented/virtual reality, speech triggers, chatbots or virtual assistants. Distributed logging and auditing capabilities further properly incentivize all stakeholders.

Arbitrary new and demanding Cloud applications (e.g., event–driven intelligent things, virtual assistants or workplace collaboration) can now base their deployment and migration decisions on in–depth benchmarking process that accurately reflects their latency requirements and treats candidate Clouds fairly.

Industry and standard bodies keep releasing protocols that suit contemporary communication network needs (e.g., CoAP and MQTT for IoT). Our work ensures that any future protocol with request–response nature plugs easily into multidimensional monitoring system and, thus has its behavior continuously recorded, compliance checked and anomalous behaviors identified.

Many conference and journal papers, already published at the time of writing, build on top of or otherwise refer to our work. These papers can be summarized by several main emerging and developing areas:

- *Novel offload infrastructures and URLLC foundations* (e.g., Hybrid Mobile Edge, server–centric PON–based fog, Two–level MAC scheduling for 5G–slicing);

- *Cloud–based low–latency applications* (e.g., Computational Fluid Dynamics, Low–latency video, hosting of demanding legacy applications);

- *Network–based evaluations* (e.g., ISM IoT mesh performance, IoT middleware performance, Amazon; S3 storage performance, PON–based DC energy–performance tradeoffs, VM migration latency–cost tradeoffs, network and storage latency attacks on trading protocols,);

- *Network–based tools* (e.g., automated Cloud auditor, network throughput and stress testing software, IoT framework, IoT simulator, performance simulation and security–requirements verification platform);

- *Cloud posture* (e.g., trust evaluation framework with compliance monitoring, network metric–based monitoring of complex applications, audit–based trust management framework);

- *Cloud network data mining* (e.g., learning from multidimensional measurements, unsupervised HCA of latency measurements, detection of spatio–temporal performance variations and event coincidences);

- *Modeling IoT devices and middleware* (e.g., reference architecture, simulation environment, security framework).

Our proposed CLAudit monitoring platform was mentioned in Devices & Networking Summit 2015 during Victor Bahl's "Cloud 2020: The Emergence of Micro Datacenters (Cloudlets) for Mobile Computing" keynote [43]. Also, several master and doctoral theses build on our work.

7.5 Research Contributions

Solutions and results covered in this thesis were published in several conference and IF–journal papers. We still publicly offer archived measurements and encourage the research community to use it for verification and further studies. The details of our original research contributions and the relevant publication record are as follows:

Chapters 4: Monitoring Methodology and 5: Data Capture Platform

- A new methodology of Cloud latency monitoring has been presented. The method is based on continuous large–scale active probing of multiple Cloud resources at multiple network protocol layers from a global network of Internet vantage points. It is lightweight, easy–to–deploy, dependable and stores little information.
- A method for multidimensional measurements capture has been devised and implemented using PlanetLab and basic–tier public Cloud provider resources.
- Cloud Latency Auditing Platform (CLAudit) platform source codebase was developed and packaged into readily–deployable monitoring solution.
- Actual real–world measurements were continuously collected using Cloud monitoring platform, visualized in a near–realtime, archived online and provided in a form of a published open datasets.

Works related to these results are:

- O. Tomanek, P. Mulinka and L. Kencl, "Multidimensional Cloud Latency Monitoring and Evaluation", *Computer Networks*, vol. 107, Elsevier, 2016, pp. 104–120.
- O. Tomanek and L. Kencl, "CLAudit: Planetary–scale Cloud Latency Auditing Platform", in *Proceedings of the 2nd International Conference on Cloud Networking (CloudNet)*. IEEE, 2013, pp. 138–146.

Chapter 6: Applications

- A mutidimensional–measurement timeseries analyses and network profiling metrics have been presented, together with revealed insights and anomalous behaviors of Cloud services.
- A new longitudinal methodology for Cloud service benchmarking has been presented that accurately reflects application requirements by leveraging preprocessed multi-dimensional measurements.
- A new MILP–based methodology for Cloud connectivity optimization has been presented that minimizes cumulative adverse network effects on Cloud–bound traffic and avoids degraded destinations by using informed–gateway decisions.
- Large–scale evaluation of Cloud service latency of two major Cloud service providers, Microsoft Azure and Amazon AWS datacenters, has been carried out through application case studies. The results revealed notable otherwise–hidden trends, events, infrastructure changes and performance differences at granularities of Cloud service providers, datacenters and resources.

Works related to these results are:

- O. Tomanek and L. Kencl, "Optimization of Cloud Connectivity using Smart–home Gateway", in *Proceedings of the 2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.
- V. Uhlir, O. Tomanek and L. Kencl, "Latency–based Benchmarking of Cloud Service Providers", in *Proceedings of the 9th International Conference on Utility and Cloud Computing (UCC)*. ACM, 2016, pp. 263–268.
- O. Tomanek, P. Mulinka and L. Kencl, "Multidimensional Cloud Latency Monitoring and Evaluation", *Computer Networks*, vol. 107, Elsevier, 2016, pp. 104–120.

Bibliography

[1] *Amazon CloudWatch*. [Online]. Available: <https://aws.amazon.com/cloudwatch/>

[2] *Amazon AWS*. [Online]. Available: <https://aws.amazon.com/>

[3] *AWS Service Health Dashboard*. [Online]. Available: <http://status.aws.amazon.com/>

[4] *Microsoft Azure*. [Online]. Available: <https://azure.microsoft.com/>

[5] *Azure IoT Edge*. [Online]. Available: <https://azure.microsoft.com/campaigns/iot-edge/>

[6] *Azure Monitor*. [Online]. Available: <https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview-azure-monitor>

[7] *Microsoft Azure status*. [Online]. Available: <http://status.azure.com>

[8] *Cacti*. [Online]. Available: <https://www.cacti.net/>

[9] "CICS Transaction Server for z/OS - Performance Guide, Version 4 Release 2," 2014, IBM SC34-7177-02. [Online]. Available: <https://www.ibm.com/support/knowledgecenter/SSGMCP_4.2.0/com.ibm.cics.ts.performance.doc/dfht3_pdf.pdf>

[10] *Cloud Harmony*. [Online]. Available: <http://cloudharmony.com/>

[11] *CloudPing*. [Online]. Available: <http://www.cloudping.info/>

[12] *CloudSleuth*. [Online]. Available: <https://www.techrepublic.com/blog/the-enterprise-cloud/cloudsleuth-independently-monitors-cloud-service-providers-performance/>

[13] *CloudWatch*. [Online]. Available: <http://www.cloudwatch.in/>

[14] *curl*. [Online]. Available: <https://curl.haxx.se/>

[15] *downrightnow*. [Online]. Available: <http://downrightnow.com/>

[16] *Amazon Echo*. [Online]. Available: <https://developer.amazon.com/echo>

[17] *AWS Greengrass*. [Online]. Available: <https://aws.amazon.com/greengrass/>

[18] *HDRHistogram*. [Online]. Available: <http://hdrhistogram.org/>

[19] *httping*. [Online]. Available: <http://www.vanheusden.com/httping/>

[20] *Indeni*. [Online]. Available: <https://indeni.com/>

[21] *Apache Kafka: A distributed streaming platform*. [Online]. Available: <http://kafka.apache.org/>

[22] *Amazon Kinesis*. [Online]. Available: <https://aws.amazon.com/kinesis/>

[23] *logz.io*. [Online]. Available: <https://logz.io/>

[24] *MRTG*. [Online]. Available: <https://oss.oetiker.ch/mrtg/>

[25] *Nagios*. [Online]. Available: <https://www.nagios.org/>

[26] *Nest*. [Online]. Available: <https://nest.com/>

[27] *Nyansa*. [Online]. Available: <https://www.nyansa.com/>

[28] *PlanetLab*. [Online]. Available: <http://planet-lab.org/>

[29] *Renesys*. [Online]. Available: <https://dyn.com/blog/category/research/>

[30] *RIPE ATLAS*. [Online]. Available: <https://atlas.ripe.net/>

[31] *Heat map of Internet connected devices*. [Online]. Available: <https://www.shodan.io/>

[32] *Splunk*. [Online]. Available: <https://www.splunk.com/>

[33] *tcping*. [Online]. Available: <http://www.linuxco.de/tcping/tcping.html>

[34] *ThousandEyes: Network Monitoring Software*. [Online]. Available: <https://www.thousandeyes.com/>

[35] *traceroute*. [Online]. Available: <http://man7.org/linux/man-pages/man8/traceroute.8.html>

[36] *Zabbix*. [Online]. Available: <https://www.zabbix.com/>

[37] J. Aikat, J. Kaur, F. D. Smith *et al.*, "Variability in TCP Round–Trip Times," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 279–284.

[38] T. Akidau, A. Balikov, K. Bekiroğlu *et al.*, "MillWheel: Fault–Tolerant Stream Processing at Internet Scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[39] M. Alizadeh, A. Greenberg, D. A. Maltz *et al.*, "Data Center TCP (DCTCP)," in *ACM SIGCOMM computer communication review*, vol. 40, no. 4. ACM, 2010, pp. 63–74.

[40] M. Alizadeh, A. Kabbani, T. Edsall *et al.*, "Less is More: Trading a Little Bandwidth for Ultra–Low Latency in the Data Center," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 19–19.

[41] M. Alizadeh, S. Yang, M. Sharif *et al.*, "pFabric: Minimal Near–Optimal Datacenter Transport," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 435–446.

[42] G. Appenzeller, I. Keslassy, and N. McKeown, *Sizing Router Buffers.* ACM, 2004, vol. 34, no. 4.

[43] V. Bahl, "Cloud 2020: The Emergence of Micro Datacenters (Cloudlets) for Mobile Computing," 2015, Devices & Networking Summit.

[44] H. Ballani, P. Costa, T. Karagiannis *et al.*, "Towards Predictable Datacenter Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[45] J. C. Bennett, K. Benson, A. Charny *et al.*, "Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 4, pp. 529–540, 2002.

[46] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* ACM, 2010, pp. 267–280.

[47] E. Bocchi, M. Mellia, and S. Sarni, "Cloud Storage Service Benchmarking: Methodologies and Experimentations," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet).* IEEE, 2014, pp. 395–400.

[48] F. Bonomi, R. Milito, J. Zhu *et al.*, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing.* ACM, 2012, pp. 13–16.

[49] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," in *LISA*, vol. 14, 2000, pp. 139–146.

[50] K.-T. Chen, Y.-C. Chang, P.-H. Tseng *et al.*, "Measuring the Latency of Cloud Gaming Systems," in *Proceedings of the 19th ACM international conference on Multimedia.* ACM, 2011, pp. 1269–1272.

[51] S. Cheshire, "Latency and the Quest for Interactivity," in *White paper commissioned by Volpe Welty Asset Management, LLC, for the Synchronous Person-to-Person Interactive Computing Environments Meeting*, 1996.

[52] S. Cheshire, *It's the Latency, Stupid*, 2016. [Online]. Available: `<http://www.stuartcheshire.org/rants/latency.html>`

[53] M. B. Chhetri, S. Chichin, Q. B. Vo *et al.*, "Smart CloudBench – Automated Performance Benchmarking of the Cloud," in *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD).* IEEE, 2013, pp. 414–421.

[54] M. Chowdhury, M. Zaharia, J. Ma *et al.*, "Managing Data Transfers in Computer Clusters with Orchestra," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 98–109.

[55] Cisco, "Design Best Practices for Latency Optimization," *Cisco white paper*, 2007.

[56] R. S. Couto, S. Secci, M. E. M. Campista *et al.*, "Latency versus Survivability in Geo–Distributed Data Center Design," in *Global Communications Conference (GLOBECOM), 2014 IEEE.* IEEE, 2014, pp. 1102–1107.

[57] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[58] C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," 2002.

[59] M. Dhawan, J. Samuel, R. Teixeira *et al.*, "Fathom: A Browser–based Network Measurement Platform," in *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012, pp. 73–86.

[60] A. B. Downey, "Evidence for Long–Tailed Distributions in the Internet," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 229–241.

[61] J. Doyle, R. Shorten, and D. O'Mahony, "Stratus: Load Balancing the Cloud for Carbon Emissions Control," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 1–1, 2013.

[62] G. Evenden, *Proj. 4-Cartographic Projections Library*, 1990. [Online]. Available: `<http://trac.osgeo.org/proj>`

[63] E. Folkerts, A. Alexandrov, K. Sachs *et al.*, "Benchmarking in the Cloud: What It Should, Can and Cannot Be," in *TPCTC*. Springer, 2012, pp. 173–188.

[64] A. Forestiero, C. Mastroianni, M. Meo *et al.*, "Hierarchical Approach for Efficient Workload Management in Geo–Distributed Data Centers," *IEEE Transactions on Green Communications and Networking*, vol. 1, no. 1, pp. 97–111, 2017.

[65] K. Gardner, S. Zbarsky, S. Doroudi *et al.*, "Reducing Latency via Redundant Requests: Exact Analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 347–360, 2015.

[66] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 5–18.

[67] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[68] Z. Hu, L. Zhu, C. Ardi *et al.*, "The Need for End–to–End Evaluation of Cloud Availability," in *PAM*. Springer, 2014, pp. 119–130.

[69] Y. Ito, H. Koga, and K. Iida, "A Bandwidth Allocation Scheme to Meet Flow Requirements in Mobile Edge Computing," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*. IEEE, 2017, pp. 1–5.

[70] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. Wiley, 1990.

[71] Y. Jiang and Y. Liu, *Stochastic Network Calculus*. Springer, 2008, vol. 1.

[72] S. Kandula, S. Sengupta, A. Greenberg *et al.*, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[73] T. Karagiannis, E. Athanasopoulos, C. Gkantsidis *et al.*, "HomeMaestro: Order from chaos in home networks," *Microsoft Research Report MSR-TR-2008-84*, 2008.

[74] R. Kay, "Pragmatic Network Latency Engineering Fundamental Facts and Analysis," *cPacket Networks, White Paper*, pp. 1–31, 2009.

[75] B. Krishnamurthy, S. Sen, Y. Zhang *et al.*, "Sketch–based Change Detection: Methods, Evaluation and Applications," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement.* ACM, 2003, pp. 234–247.

[76] J. F. Kurose and K. W. Ross, "Computer Networking: A Top–Down Approach," *Addison Wesley*, vol. 4, p. 8, 2007.

[77] M. Kwon, "A Tutorial on Network Latency and its Measurements," *IGI Global*, 2015.

[78] Z. Lai, Y. Cui, M. Li *et al.*, "TailCutter: Wisely Cutting Tail Latency in Cloud CDN under Cost Constraints," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications.* IEEE, 2016, pp. 1–9.

[79] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network–Wide Traffic Anomalies," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[80] J.-Y. Le Boudec, "Application of Network Calculus to Guaranteed Service Networks," *IEEE Transactions on Information theory*, vol. 44, no. 3, pp. 1087–1096, 1998.

[81] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems.* EPFL Press, 2010.

[82] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer Science & Business Media, 2001, vol. 2050.

[83] C. Lee, K. Jang, and S. Moon, "Reviving Delay–based TCP for Data Centers," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures and protocols for computer communication.* ACM, 2012, pp. 111–112.

[84] A. Lenk, M. Menzel, J. Lipsky *et al.*, "What Are You Paying For ? Performance Benchmarking for Infrastructure–as–a–Service Offerings," in *2011 IEEE International Conference on Cloud Computing (CLOUD).* IEEE, 2011, pp. 484–491.

[85] É. Leverett, R. Clayton, and R. Anderson, "Standardisation and Certification of the Internet of Things," 2017.

[86] A. Li, X. Yang, S. Kandula *et al.*, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* ACM, 2010, pp. 1–14.

[87] J. Li, N. K. Sharma, D. R. Ports *et al.*, "Tales of the Tail: Hardware, OS and Application–Level Sources of Tail Latency," in *Proceedings of the ACM Symposium on Cloud Computing.* ACM, 2014, pp. 1–14.

[88] Z. Li, M. Liang, L. O'brien *et al.*, "The Cloud's Cloudy Moment: A Systematic Survey of Public Cloud Service Outage," *arXiv preprint arXiv:1312.6485*, 2013.

[89] H. V. Madhyastha, T. Anderson, A. Krishnamurthy *et al.*, "A Structural Approach to Latency Prediction," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 99–104.

[90] G. Mahlknecht, "Greg's Cable Map," *Retrieved Sep*, vol. 12, 2011. [Online]. Available: `<http://www.cablemap.info/>`

[91] M. Marshak and H. Levy, "Evaluating Web User Perceived Latency using Server Side Measurements," *Computer Communications*, vol. 26, no. 8, pp. 872–887, 2003.

[92] M. M. S. Maswood and D. Medhi, "Optimal Connectivity to Cloud Data Centers," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*. IEEE, 2017, pp. 1–6.

[93] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[94] M. Menth, R. Martin, and J. Charzinski, "Capacity Overprovisioning for Networks with Resilience Requirements," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 87–98.

[95] M. Menzel and R. Ranjan, "CloudGenius: Decision Support for Web Server Cloud Migration," in *Proceedings of the 21st International conference on World Wide Web*. ACM, 2012, pp. 979–988.

[96] R. Minnear, "Latency: The Achilles Heel of Cloud Computing," *Cloud Computing Journal*, 2011.

[97] T. Mizrahi and Y. Moses, "On the Behavior of Network Delay in the Cloud," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 875–876.

[98] A. Mukherjee, "On the Dynamics and Significance of Low Frequency Components of Internet Load," *Technical Reports (CIS)*, p. 300, 1992.

[99] P. Mulinka and L. Kencl, "Learning from Cloud Latency Measurements," in *2015 IEEE International Conference on Communication Workshop (ICCW)*. IEEE, 2015, pp. 1895–1901.

[100] M. Naldi, "ICMP–based Third–party Estimation of Cloud Availability," *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 6, no. 1, pp. 11–18, 2017.

[101] J. Osborne, "Notes on the Use of Data Transformations," *Practical assessment, research and evaluation*, vol. 9, no. 1, pp. 42–50, 2005.

[102] J. Padhye, V. Firoiu, D. Towsley *et al.*, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 303–314, 1998.

[103] A. Pathak, A. Wang, C. Huang *et al.*, "Measuring and Evaluating TCP Splitting for Cloud Services," in *PAM*, vol. 10. Springer, 2010, pp. 41–50.

[104] V. Paxson, "Empirically Derived Analytic Models of Wide–Area TCP Connections," *IEEE/ACM Transactions on Networking (TON)*, vol. 2, no. 4, pp. 316–336, 1994.

[105] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking (TON)*, vol. 3, no. 3, pp. 226–244, 1995.

[106] V. E. Paxson, "Measurements and Analysis of End–to–End Internet Dynamics," Ph.D. dissertation, University of California, Berkeley, 1997.

[107] J. Perry, A. Ousterhout, H. Balakrishnan *et al.*, "Fastpass: A Centralized Zero–Queue Datacenter Network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[108] I. Rish, M. Brodie, N. Odintsova *et al.*, "Real–Time Problem Determination in Distributed Systems using Active Probing," in *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1. IEEE, 2004, pp. 133–146.

[109] R. Rocha, *The IoT Architecture at the Edge*. [Online]. Available: `<https://www.iotcentral.io/blog/the-iot-architecture-at-the-edge>`

[110] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End–to–End Arguments in System Design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, 1984.

[111] M. Satyanarayanan, P. Bahl, R. Caceres *et al.*, "The Case for VM–based Cloudlets in Mobile Computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.

[112] S. R. Smoot and N. K. Tan, *Private Cloud Computing: Consolidation, Virtualization and Service–Oriented Infrastructure*. Elsevier, 2012.

[113] N. Spring, L. Peterson, A. Bavier *et al.*, "Using PlanetLab for Network Research: Myths, Realities and Best Practices," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 17–24, 2006.

[114] D. Strom and J. F. van der Zwet, "Truth and Lies about Latency in the Cloud," *InterxionTM white paper*, 2012.

[115] G. Sun, H. Yu, V. Anand *et al.*, "Optimal Provisioning for Virtual Network Request in Cloud–based Data Centers," *Photonic Network Communications*, vol. 24, no. 2, pp. 118–131, 2012.

[116] G. Tene, "How Not to Measure Latency," *Low Latency Summit*, 2013.

[117] O. Tomanek and L. Kencl, "CLAudit: Planetary–Scale Cloud Latency Auditing Platform," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*. IEEE, 2013, pp. 138–146.

[118] O. Tomanek and L. Kencl, "Security and Privacy of Using AllJoyn IoT Framework at Home and Beyond," in *2016 2nd International Conference on Intelligent Green Building and Smart Grid (IGBSG)*. IEEE, 2016, pp. 1–6.

[119] O. Tomanek and L. Kencl, "Optimization of Cloud Connectivity using a Smart–Home Gateway," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[120] O. Tomanek, P. Mulinka, and L. Kencl, "Multidimensional Cloud Latency Monitoring and Evaluation," *Computer Networks*, vol. 107, pp. 104–120, 2016.

[121] A. N. Toosi, C. Qu, M. D. de Assunção *et al.*, "Renewable–Aware Geographical Load Balancing of Web Applications for Sustainable Data Centers," *Journal of Network and Computer Applications*, vol. 83, pp. 155–168, 2017.

[122] R. Tripathi, S. Vignesh, and V. Tamarapalli, "Optimizing Green Energy, Cost and Availability in Distributed Data Centers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 500–503, 2017.

[123] Y. Tsang, M. Coates, and R. D. Nowak, "Network Delay Tomography," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2125–2136, 2003.

[124] V. Uhlir, O. Tomanek, and L. Kencl, "Latency–based Benchmarking of Cloud Service Providers," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*. ACM, 2016, pp. 263–268.

[125] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline–Aware Datacenter TCP (D2TCP)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115–126, 2012.

[126] T. Verbelen, P. Simoens, F. De Turck *et al.*, "Cloudlets: Bringing the Cloud to the Mobile User," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 2012, pp. 29–36.

[127] O. Vondrous, P. Macejko, and Z. Kocur, "FlowPing – The New Tool for Throughput and Stress Testing," *Advances in Electrical and Electronic Engineering*, vol. 13, no. 5, p. 516, 2015.

[128] A. Vulimiri, P. B. Godfrey, R. Mittal *et al.*, "Low Latency via Redundancy," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 283–294.

[129] Y. A. Wang, C. Huang, J. Li *et al.*, "Estimating the Performance of Hypothetical Cloud Service Deployments: A Measurement–based Approach," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2372–2380.

[130] C. Wilson, H. Ballani, T. Karagiannis *et al.*, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 50–61.

[131] W. Xu and J. Rexford, "MIRO: Multi-Path Interdomain Routing," in *Proceedings of the ACM SIGCOMM 2006 conference on Applications, technologies, architectures and protocols for computer communication*. ACM, 2006, pp. 171–182.

[132] Y. Xu, Z. Musgrave, B. Noble *et al.*, "Bobtail: Avoiding Long Tails in the Cloud," in *NSDI*, vol. 13, 2013, pp. 329–342.

[133] M. Zhang, C. Zhang, V. S. Pai *et al.*, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide–Area Services," in *OSDI*, vol. 4, 2004, pp. 12–12.

[134] A. C. Zhou, B. He, X. Cheng *et al.*, "A Declarative Optimization Engine for Resource Provisioning of Scientific Workflows in Geo–Distributed Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 647–661, 2017.

[135] Y. Zhu, H. Eran, D. Firestone *et al.*, "Congestion Control for Large–Scale RDMA Deployments," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 523–536.

Appendix A

List of Topical Publications

Publication I is a direct output of project VII (see Appendix. C). Publication II is a direct output of project III. Publication III is a direct output of project VII. Publication IV is the aggregate output of projects IX, X and XI. All authors contributed equally to the respective listed papers.

IF–journal papers

I) O. Tomanek, P. Mulinka and L. Kencl, "Multidimensional Cloud Latency Monitoring and Evaluation", *Computer Networks*, vol. 107, Elsevier, 2016, pp. 104–120. [Online]. Available: `<http://www.sciencedirect.com/science/article/pii/S138912861630192X>`

Citations except self–citations: 0 (WoS), 1 (Scopus), 6 (Google Scholar).

Scopus and WoS–indexed conference papers

II) O. Tomanek and L. Kencl, "Optimization of Cloud Connectivity using Smart–home Gateway", in *Proceedings of the 2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

Citations except self–citations: 0 (WoS), 0 (Scopus), 0 (Google Scholar).

III) V. Uhlir, O. Tomanek and L. Kencl, "Latency–based Benchmarking of Cloud Service Providers", in *Proceedings of the 9th International Conference on Utility and Cloud Computing (UCC)*. ACM, 2016, pp. 263–268.

Citations except self–citations: 0 (WoS), 2 (Scopus), 3 (Google Scholar).

IV) O. Tomanek and L. Kencl, "CLAudit: Planetary–scale Cloud Latency Auditing Platform", in *Proceedings of the 2nd International Conference on Cloud Networking (CloudNet)*. IEEE, 2013, pp. 138–146.
Citations except self–citations: 5 (WoS), 6 (Scopus), 7 (Google Scholar).

Appendix B

List of Other Publications

Publication I is a direct output of project VII (see Appendix. C). Publication II is the aggregate output of projects IV, V and VI. All authors contributed equally to the respective listed papers.

Scopus and WoS–indexed conference papers

I) Z. Kouba, O. Tomanek and L. Kencl, "Evaluation of Datacenter Network Topology Influence on Hadoop MapReduce Performance", in *Proceedings of the 5th IEEE International Conference on Cloud Networking (CloudNet).* IEEE, 2016, pp. 95–100.
Citations except self–citations: 0 (WoS), 0 (Scopus), 1 (Google Scholar)

II) O. Tomanek and L. Kencl, "Security and Privacy of Using AllJoyn IoT Framework at Home and Beyond", in *Proceedings of the 2nd International Conference on Intelligent Green Building and Smart Grid (IGBSG).* IEEE, 2016, pp. 1–6.
Citations except self–citations: 0 (WoS), 1 (Scopus), 4 (Google Scholar).

Appendix C

List of Projects

Contributing to this thesis was work conducted in a scope of projects I, III, VII, IX, X and XI. Projects II, IV, V, VI and VIII were independently conducted industry collaborations with partner research organizations.

I) P. Mulinka, J. Klemsa, O. Tomanek and L. Kencl, "Privacy Protection and Machine–Learning Utilization of IoT Data in Cloud", SGS18/077/OHK3/1T/13, Co–investigator role, 1/2018 – 1/2019, Czech Technical University in Prague

II) O. Tomanek et al., "OTA Update Scalability and Capacity Planning", Principal Investigator role, 2/2017 – 8/2017, Electrolux – CTU Prague Technology Centre

III) P. Mulinka, O. Tomanek, J. Klemsa and L. Kencl, "Smart–home IoT and Cloud Telemetry Datamining", SGS17/091/OHK3/1T/13, Co–investigator role, 1/2017 – 1/2018, Czech Technical University in Prague

IV) O. Tomanek et al., "AllJoyn Security as of Release 16.10", Principal Investigator role, 7/2016 – 1/2017, Electrolux – CTU Prague Technology Centre

V) O. Tomanek et al., "AllJoyn Security as of Version 16.04", Principal Investigator role, 11/2015 – 7/2016, Electrolux – CTU Prague Technology Centre

VI) O. Tomanek et al., "Securing AllJoyn Devices and Interfaces", Principal Investigator role, 5/2015 – 11/2015, Electrolux – CTU Prague Technology Centre

VII) O. Tomanek, P. Mulinka, Z. Kouba, V. Uhlir, E. Marku and L. Kencl, "Cloud Performance Analysis and Improvement", SGS15/153/OHK3/2T/13, Principal Investigator role, 1/2015 – 1/2017, Czech Technical University in Prague

VIII) O. Tomanek et al., “Cloud Security – Data Services Delivery”, Cisco International Internship Program, Software Developer role, 7/2013 – 8/2014, Cisco Systems, San Jose, CA, USA

IX) L. Kencl and O. Tomanek, “Measurement and Analysis of Latency for Cloud Computing Network Optimization”, grant no. 497/2013, Co–investigator role, 1/2013 – 1/2014 and 1/2015 – 7/2015, CESNET

X) O. Tomanek and L. Kencl, “Windows Azure Academic Research Grant,” Principal Investigator role, 1/2013 – 1/2015, Microsoft Research

XI) O. Tomanek, J. Stanek, P. Mulinka and L. Kencl, “Methods Enhancing Work with Cloud Data”, SGS13/139/OHK3/2T/13, Principal Investigator role, 1/2013 – 1/2015, Czech Technical University in Prague

Appendix D

Other Results

- O. Tomanek, "CLAudit data sets," early–2013 to late–2017. [Online]. Available: <http://claudit.feld.cvut.cz/data.php>
- O. Tomanek, "CLAudit platform source codebase," 2015. Property of the partner research organization CESNET.