School of Electronic Engineering and Computer Science

Trade-offs in Edge Computing

PhD Thesis

Yousef Amar

Primary Supervisor: Gareth Tyson
Secondary Supervisor: Gianni Antichi
Independent Assessor: Steve Uhlig

2019-12-31

Abstract

Internet-connected devices have become ubiquitous and we are generating more personal data than ever before. Recent lines of research have investigated edge computing as a means to process this data closer to the source. Doing so would protect users' privacy, save bandwidth, and lower latencies.

We examine edge computing at several levels to determine where the trade-offs lie and how we can leverage the characteristics of edge computing to our advantage. Through the lens of privacy, performance, cost, and utility we tackle the issues of optimal job distribution, access control and discovery, and peer to peer network topologies.

Our research empirically demonstrates the advantages of distributing computation at the network edge for a variety of use cases. We build several systems in this context and throughly evaluate them. The ultimate goal of our work is to push for a more private and efficient internet ecosystem.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research context

Over the past decade alone, the world has seen an explosion in the quantity of personal data people produce daily [59]. This can be attributed not just to a larger proportion of our lives moving online, but also through the proliferation of ubiquitous sensing through wearable, mobile, and Internet of Things (IoT) devices.

These data sources generate a rapidly increasing amount of data at the network edge [5]. To data processors, this data is a treasure trove for analytics, be it for personalised services, targeted advertising, research, or any number of other purposes.

As storage space for our online and social media data cheapens, this quantity continues to increase exponentially. Alongside this surge, concerns over privacy, trust, and security are expressed more and more as different parties attempt to take advantage of this rich assortment of data.

While this gold rush has been underway, best practices for secure storage and processing of data, and the legislation that enforces these, have been lagging behind. It is therefore not surprising when we read about yet another data breach [57] that leaks the personal data of millions of people and puts them at risk.

Our devices becoming more powerful does present an important opportunity however; server-side processing our data is no longer the only option for analytics. It is now viable to shift the majority of analytics to the network edge, towards clients. This not only has privacy benefits, but can also improve application performance.

At the same time, there is a distinct need to integrate an awareness of privacy into the systems that handle our personal data in such a way that risk is minimised while

at the same time utility and performance are preserved. Systems that are privacy-aware by design can operate in a privacy-preserving manner, allowing data processors to focus on the processing and data subjects to not have to worry about risks to their privacy stemming from careless processing and/or bad systems design.

It is becoming increasingly critical that we rebalance the distribution of power between data subjects and data controllers, while at the same time not impinging on the main advantages that full control of data provides to controllers. The various fragmentary attempts at addressing this need, many of which are discussed in this thesis, are generally fraught with the shortcomings of centralisation or implicitly overexpose personal data to third parties.

Equally important is understanding the unavoidable trade-off that exists between implementing privacy-preserving mechanisms and incurring performance hits. This holds especially true for real-time mechanisms, such as trusted computation [49] as — unlike data obfuscation and other data-centric mechanism — this cannot be done offline or as part of a pipeline. Acceptable performance must be maintained in order for any privacy-preserving mechanisms to be considered practical.

In this work, we introduce, implement, and evaluate different methods for making certain systems privacy-preserving, as well as the trade-offs in utility and performance associated with these.

1.2 Motivation

With the prevalence of cloud computing and — more recently — edge computing, personal data is being sent potentially anywhere in the world. In most cases, where data is sent to is decided such that performance metrics (e.g. latency and throughput) are optimised [84]. Privacy has historically taken a back seat to user experience, which performance affects more directly.

Similarly, access control systems to personal data are coarse and opaque. At best, access is "all-or-nothing" or allows superficial fixed throttling such as rate limiting. Models used in social media APIs or smartphone sensor interfaces for example are limited in granularity.

Very little research has been done into enhancing access control and job distribution systems to be privacy preserving. This gap in our knowledge is both as a result of the relative novelty of this research area, as well as the current lack of incentives for service providers to take privacy into account **[Citation Needed]** clarify

Edge computing is an obvious solution to this, but fundamentally, the decision of which edge device should execute units of computation, and how data is shared across these devices, are hard problems. Further, there are cost and performance incentives attached to solving these problems properly, making them that much more important.

As questions of privacy are moving further and further into the forefront of consumers and legislators' concerns, it is critical that we understand exactly what implications current systems have regarding privacy risks, how to address these, and what effect they have on other factors such as performance and utility.

We claim that well designed systems can preserve privacy at little to no extra temporal, monetary, or qualitative cost. Where there is a cost, it can be shown that the trade-off is empirically justifiable. Fundamentally, the benefits of this research is clear: more intrinsic privacy in how our personal data is handled will result in less risk overall that adverse parties can use this data to draw unwanted inferences, and the cost and performance benefits of these solutions make them commercially enticing.

1.3 Research objectives

The primary focus of this research is to implement and evaluate novel systems for computation at the network edge that optimise for privacy and performance. These systems allow us to control and attenuate the flow of data from one node to another with respect to performance considerations. We consider this especially in the edge computing and IoT context with high frequency data (such as smartphone or IoT sensor data) as this context is both increasingly prevalent yet widely uncharted in its relative novelty. We also consider other use cases with low latency requirements and large volumes of data, such as multi-user virtual environments.

The theme around which this thesis revolves is a question that will come up over and over again: *Where do we execute computations?* We further split this into three focal points:

1. **The Computation** — How can augment job distribution systems such that fewer nodes consume data at no cost to performance and utility?
2. **The Data** — How can we take this further by augmenting distributed discovery and access control systems to instill them with a built-in notion of privacy?
3. **The Network** — How can we scalably interconnect edge nodes in such a way that we minimise server involvement and costs?

The first question focuses on the computation and distributing it in such a way that we optimise for various factors. The second question focuses on data and the systems for interfacing with it, and the main contributions lie in designing and implementing these such that they are privacy-preserving. The third question focuses on the underlying networks and computing optimal topologies for these to make them not only viable alternative solutions to traditional approaches, but preferred ones. Finally, we must evaluate practical implementations of these approaches and make the case for their feasibility. While these three topics are quite interlinked, this thesis is split up in a way that roughly divides them into distinct chapters.

1.4 Summary of contributions

This section summarises the contributions made in the course of this PhD and lists all accepted publications relating to this work.

1.4.1 Summary of publications

2019

Towards Cheap Scalable Browser Multiplayer

In Conference on Games (CoG) IEEE

> In this work, we introduce and evaluate a P2P-based method and library that aims to minimse running costs and development overhead for independent, multiplayer, browser games.

Zest: REST over ZeroMQ

In Proceedings of the 3rd Workshop on Security, Privacy and Trust in the Internet of Things, in conjunction with IEEE PERCOM

> In this paper we introduce, Zest (REST over ZeroMQ), a middleware technology in support of an Internet of Things (IoT).

2018

Providing Occupancy as a Service with Databox

In Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing (pp. 29-34). ACM

> In this paper we present Occupancy-as-a-Service (OaaS) implemented as an app on Databox.

An Information-Theoretic Approach to Time-Series Data Privacy

In Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems (p. 3). ACM

> In this paper, we present a system for tuning a data consumer's access to personal data based on real-time privacy metrics.

Building Accountability into the Internet of Things: The IoT Databox Model

Journal of Reliable Intelligent Environments (pp. 1-17) Springer

> This paper outlines the IoT Databox model as a means of making the Internet of Things (IoT) accountable to individuals.

2017

Balanced Message Distribution in Distributed Message Handling Systems

US Patent (serial number: 15/794440)

> This patent describes a combination of methodologies for arriving at near-optimal message distribution decisions in distributed messaging systems under specific constraints.

Route-based Authorization and Discovery for Personal Data

In the 11th EuroSys Doctoral Workshop

> When faced with systems in which third party components need to advertise the availability of data they gather, while other such components need to access it, solutions for delegated authorisation and discovery APIs for interoperability are needed. This work explores possible solutions, and converges on a testable implementation.

2016

Personal Data Management with the Databox: What's Inside the Box?

In Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking (pp. 49-54) ACM

A more detailed look at the Databox as it stood; a collection of physical and cloudhosted software components that provide for an individual data subject to manage, log and audit access to their data by other parties.

Privacy-Aware Infrastructure for Managing Personal Data

In Proceedings of the 2016 ACM SIGCOMM Conference (pp. 571-572) ACM

A poster abstract giving an overview of Databox systems as they stood with a stronger focus on arbiter interactions.

1.4.2 Research outcomes

- Design, simulation, implementation, and evaluation of a novel job distribution system for optimising completion time over edge devices
- Design, simulation, and implementation of a dynamic data privacy "filter"
- Design, simulation, implementation, and evaluation of a privacy-aware extension of our job distribution system
- Design an implementation of a phone sensor streaming client and server pair
- Implementation and release of a toolkit home network analysis
- Analysis of home network traces and fingerprinting of consumer home IoT devices
- Demonstrating methods of identifying, monitoring, and controlling these devices
- Design, implementation, and evaluation of a prototype of private home data analytics platform Databox, as well as contributions to later versions
- Design, implementation, and comprehensive evaluation of a secure, distributed authorisation system (applied to Databox) with privacy-awareness

- Design, implementation, and comprehensive evaluation of a secure, distributed service/data discovery system (applied to Databox) with privacy-awareness
- Design, simulation, implementation, and evaluation of a peer network topology system targeted at large-scale multi-user virtual environments
- Design and implementation of a general-purpose browser framework for orchestrating complex P2P networks
- Implementation of other common P2P models and solutions within our framework, and systematic comparison to ours
- incomplete

1.5 Thesis structure

In this section, we briefly lay out the structure of this document.

We begin with a high-level overview of the background on our research area. We limit this chapter to general concepts and motivational discourse, such as edge computing as a paradigm, decentralisation, the kinds of use cases we look at, and how privacy is measured. We take a more detailed look at specific literature in the topic chapters that follow. In §1.3 we divided the scope of our research into three angles of attack. These angles roughly correspond to the three topic chapters that make up this thesis.

In the first, we dive deep into job distribution, and we present our solutions for optimising it in the context of edge computing. Then we present and evaluate our system for building privacy contexts, and we show how we apply this system to our job distribution solutions.

In the second, we go a step further and examine edge computing in the home in an IoT context. We scrutinise smart home network traffic in detail, and build applicable access control and discovery systems, which we also evaluate. To do this, we revisit our solutions from the previous chapter.

In the third, we take a closer look at networks and communication at the edge. We examine common architectures and present our own that optimises for characteristic use cases. We also build a framework for deploying different architectures and we use this to evaluate our own solution and compare it with others.

Finally, we bring everything together and discuss the impact of this work and future developments.

Chapter 2

Background

In this chapter we explore and review high-level concepts and literature pertinent to the research areas of this PhD project.

2.1 Edge Computing

As IoT devices proliferate and mobile devices become ubiquitous, they generate a rapidly increasing amount of data at the network edge [5]. Services that make use of this data often require low and stable latency. Edge computing has become a popular approach to fulfilling this requirement, as pushing computation to the network edge shifts processing closer to the sources and destinations of data, thus circumventing the overhead of processing and transferring data in the network core [84].

With edge computing, a variety of challenges also arise. One such challenge is to contend with fewer resources on edge devices. To address this challenge, a common approach is to distribute computation over a transient cloud of edge devices such as mobile phones [42, 69]. However, deciding which device should receive which job, in order to minimize the overall response time[1], is not trivial, let alone more abstract factors such as privacy risk. Existing job distribution approaches make assumptions about the execution context that do not apply here, such as homogeneity between datacanter servers, CPU cores, or GPU kernels. Our contributions seek to address this problem.

We make three key contextual observations that motivate our work. First, unlike a cluster of servers in a traditional cloud, the edge devices are most likely not homogeneous in terms of capabilities and resources, and as a result, execution capacity. Certain devices

[1]Response time is defined as queuing delay plus actual service time.

may indeed be incapable of executing certain jobs simply due to the lack of necessary hardware (e.g., GPU for highly-parallelizable image processing). Similarly, these devices may be running many extraneous processes that affect the availability of resources over time and thus the time to execute other jobs.

Second, querying the metadata and states of these edge devices over the network takes at least as long as pushing jobs to them, thus incurring significant overhead and latency. As a result, operations to distribute jobs on these devices necessitate the ability to work with stale device information.

Finally, it is difficult to query device resources and map them to estimates of job response time or privacy risk, because the potential diversity among jobs and devices makes this mapping unreliable. Challenges in synchronization and coordination arise.

In edge computing, context-awareness on the edge is a well explored topic, and recent work has been extensively surveyed [103]. The problem of adding context to job distribution on the edge has been tackled through Transient Context-Aware Clouds (TCAC) [110], which do not guarantee near-optimal job response time but still achieve some reduced job response time, because their job distribution considers device context such as location and resources. However, such distribution is not generalizable, and context changes over time are not considered.

Other deciding factors may include the preservation of IP

2.2 Centralisation vs decentralisation: a brief history

In the sixties, when a computer with the most basic capabilities could fill an entire room, all computation was done on mainframes. People would connect to these through terminals that did nothing but interface with the mainframes, and jobs would be scheduled and run on the mainframes with shared resources. Computing resources and performance were major bottlenecks, and security was barely a concern at this point in time. This was akin to having processing concentrated server-side with minimal clients interfacing with the servers.

Once desktop computers started becoming practical and affordable in the seventies, we saw a shift away from the centralisation of mainframes, and programs were instead run locally. Once the internet started to gain momentum, it looked much more decentralised and federated than it does today. In its early days, when IPv4 addresses were thought

to never run out, and NATs were not needed, webpages were being served from the same machines that request them.

When Napster was released in 1999, many Peer-to-Peer (P2P) protocols and networks followed, and we also started seeing the internet being used for applications such as VoIP. At the same time, certain web pages were gaining a lot of traction and introducing information asymmetry on the internet. Websites were, and still are, hosted on dedicated servers that have the resources to handle larger volumes of traffic.

As servers and storage became cheaper at scale, many web service providers have adopted the model of server-side rendering, leaving client devices to mostly display static pages that would be considered simple by today's standards. Through competition, client devices and browser quickly caught up in resources and capabilities however, so modern web apps have shifted back to Client-Side Rendering (CSR), through Single-Page Applications (SPAs) and Progressive Web Apps (PWAs), in order to improve user experience and lower server costs.

More recently, there has been a push to reduce overheads and improve scalability by attempting to split application logic into microservices and more discrete units of logic [48]. Server-side deployment has gone from virtual machines, to containers which share an OS, to lambdas (in a serverless context) which share a runtime [51]. The additional granularity has the advantage lowering server costs as you only pay for exactly what you use.

That being said, each paradigm still requires elaborate and labour-intensive orchestration and management of infrastructure at scale. We can lower costs even further by shifting as much server-side logic as possible to the clients and at the same time simplify the development overhead using appropriate abstractions and as all development becomes client-side development.

The current state-of-the-art in serverless computing are Cloudflare Workers [11] which, in addition to runtime optimisations, run on servers as close to the user as possible. Even here, the pull of distribution and decentralisation is evident and strong, making a resurgence whenever it is possible, simply because the advantages are too large to ignore. To this day, new developments that utilise distributed technology, such as Distributed Hash Tables (DHTs) or blockchains, continue to create systems with advantage in resilience, privacy, scalability, cost, and more.

2.3 Modern use cases

2.3.1 Computation and jobs

A "job" in the context of computation can have a wide range of definitions. For our purposes, we define a job as an invocation of a standalone function on distributed devices. This can be analogous to functions in map-reduce-like distributed computation architectures, but also to lambdas in the serverless computing context which we discuss in more detail in §3.

Computation can also include longer-lived processes, as opposed to short-lived function invocations. We consider these kinds as well, especially in the context of streaming data, as well as distributed interactive simulations.

2.3.2 IoT devices

At the most abstract level, wherever there is a *consumer* of data, there is also a *producer*. A consumer can receive a variety of different data streams, and a producer can also be a consumer and vice versa. These nodes — producers, consumers, or both — can exist in complex networks of streams. In addition to the inferences a consumer can make based on any one stream, a consumer can also draw additional inferences through correlating streams.

Practically, these producers and consumers can manifest themselves in many different ways, such as physical devices in a network, or logically distinct components in a system on a single device, or a hybrid of the two. We describe our solutions using higher-level terms that can then be mapped to lower-level instances of data controller networks.

To implement and evaluate our solutions, we focus on the edge computing context. Large proportion of our data being generated by home IoT devices and sensors near the edge, and applications that make use of this data often require low latency, such as home automation use cases.

Pushing computation to the edge as opposed to pushing data to to the computation in the cloud also yields advantages through increasing privacy. Not only is the data that is sent to the cloud limited to only the results of a computation, but by limiting the data that is sent to the cloud, the potential effect of breaches is reduced a lot.

2.3.3 Personal data analytics

Privacy in the context of personal data is a well explored topic. Usually privacy and anonymity are used interchangeably, but there are some very distinct differences between the two. The demand for privacy exists despite anonymity, and is indeed more pronounced when individuals are not anonymous.

Specifically when dealing with statistical data when analysing trends for market research for instance, the study of Differential Privacy [31, p. 2, 88] mostly deals with deanonymization, which is not as subjective as privacy in general; you are either identifiable or you are not. Even in that case, privacy leakage is still rampant despite so-called anonymity [28].

There have been some interesting approaches to solving these problems, for example through homomorphic encryption, which would allow processing encrypted data and thus never exposing the underlying information [107], however the practicality of this is debatable [94] and yet to be proven commercially.

Furthermore, privacy aside, there have been relatively few attempts at solving the issue of providing standard interfaces to different kinds of data with the data subject in control. Most of these few attempts fail to take the user side into account that would prevent mass adoption due to perceived risks and incomplete consumer understanding. Achieving a healthy balance between accurate and beneficial analytics, while respecting the individuals' privacy, remains an unresolved challenge.

Software such as Mydex [93] attempt to address some of these issues by way of Personal Data Stores, allowing subject to retain "ownership" of their data and provide it to 3rd parties on demand [100]. While this does offer some degree of accountability, most privacy concerns are still intact as processing is done remotely after all the needed data is transferred to that party – not to mention not providing any real incentive for either subject or controller to use.

There are also attempts to enable centralised research on diverse data, for example BRISSKit on health and medical data [12], that forego the advantages of data subject ownership. They may allow exceedingly fruitful analysis, but that is in turn mitigated by privacy issues.

Similarly, approaches that involve controlling access to encrypted personal data in the cloud, such as Sieve [122], are prey to the same pitfalls as those of homomorphic encryption in the cloud. It can be argued that these are even less secure as they involve decrypting data to provide to processors.

Approaches such as openPDS [91] and systems such as Dataware [6] do go one step further in moving the processing to the data as opposed to the data to the processing, yet face the very same technical hurdles that must be address and studied in practice especially.

It is clear to see however, that the status quo tends to involve handing over your personal data in some form to a third party for whatever purpose. The fundamental architectural challenges that this work tackles are based on an approach that solves the aforementioned issues of privacy and usability. This is the paradigm of the Databox, where the processing is done subject-side in a structured manner. This approach is summarised in the following section.

2.4 Privacy metrics

Privacy can be defined in a range of subjective ways. Sometimes it is used interchangeably with anonymity, but there are crucial differences. The goal of this section is to explore the different ways in which it can be defined, and how we define it for the purpose of our research.

2.4.1 General

Privacy in the context of personal data is a well explored topic. It is not the goal of this work to develop new metrics, but rather to apply and evaluate existing metrics to the systems we develop.

Often privacy and anonymity are used interchangeably, but there are some very distinct differences between the two. The demand for privacy exists despite anonymity, and is indeed more pronounced when individuals are not anonymous.

Dalenius first coined the term *quasi-identifier* in 1986 [26] and since then, a number of seminal publications have dealt with the process of identifying individuals by making inferences from data that may not contain any explicit identifiers (such as a UID).

Famous examples include the ability to uniquely identify 87% of the population of the United States by combining gender, birth date, and post code information [116], as well as the deanonymization of the Netflix Prize Dataset by combining it with public IMDB data [96].

While deanonymization relies on the linkage of data to explicit identifiers, more privacy-centric methods focus on making it more difficult to connect sensitive attributes

to individuals.

Recent, comprehensive survey papers describe an extensive range of metrics for a vast array of different purposes [37] and organise these into taxonomies [121].

While many of these provide average risk measurements over a given dataset, some have been repurposed to provide "one-symbol information", or the marginal mutual information from the appending of an additional record [9, 8].

The following is a description of a number of metrics suitable to our method. We divide these by output measure into two categories based on Wagner and Eckhoff's taxonomy [121].

2.4.2 Similarity/diversity

K-anonymity [115] is an exceedingly prevalent privacy measure. To quote the original paper, "A release provides k-anonymity protection if the information for each person contained in the release cannot be distinguished from at least k-1 individuals whose information also appears in the release". Explicit identifiers are completely suppressed and quasi-identifiers generalised. Similarly, rows in time series data can form equivalence classes after microaggregation based on a quasi-identifier column, where the smallest cluster has k rows.

L-diversity [83] is an extension of k-anonymity that additionally requires that sensitive attributes are well-represented in each equivalence class (for various definitions of "well represented"). It is therefore less susceptible to homogeneity attacks and background knowledge attacks. L-diverse data is by definition at least l-anonymous.

T-closeness [79] goes yet another step beyond and takes account of the distance between the distribution of sensitive attributes in any single equivalence class, to the distribution of sensitive attributes across the whole dataset. The distance measure is arbitrary, though the original paper uses Earth Mover's Distance. T-closeness for the whole dataset is the maximum of t-closeness for each equivalence class. This addresses potential attacks on l-diversity such as skewness attacks.

2.4.3 Information gain/loss

While pure entropy is an average value over a distribution, we want a marginal measure for every symbol. This is where information **surprisal** becomes useful. Surprisal is also known as self-information, however as this term is sometimes used interchangeably with

entropy, we will refer to it as surprisal throughout this paper. As a measure, it has been used in the past to, for example, measure information gain from the attributes of public social media profiles [19]. When sampling a variable, surprisal is a measure (in information-entropic bits) of uncertainty associated with sampling this variable — the negative logarithm of the probability of a sample.

Finally, many inferences can be made just from identifying patterns in time series data. A simple example is inferring location from temperature data, while a more advanced example is identifying what you watch on TV from smart meter data [44]. The Pearson correlation coefficient has been used in the past to compare smart meter data before and after anonymization [70] as opposed to other common distance measures such as KL-divergence [63].

Similarly, we can can cross-correlate data with itself shifted by varying time lags: **autocorrelation**. This allows us to detect seasonality and patterns in time series data, and on doing so, suppress, shift, or perturb the output.

Chapter 3

The job distribution decision

3.1 Overview

In this context, jobs are arbitrary instantiations of computation that require a certain amount of work to be executed in its entirety on a particular data processor. Consolidate: host, device, processor, compute node, consumer As mentioned in § 1.1, these data processors are heterogeneous edge devices such as mobile and IoT devices, as well as cloud servers. This also means that for identical jobs, different consumers may have different job completion times. This context is reflected in our simulation setup and implementations.

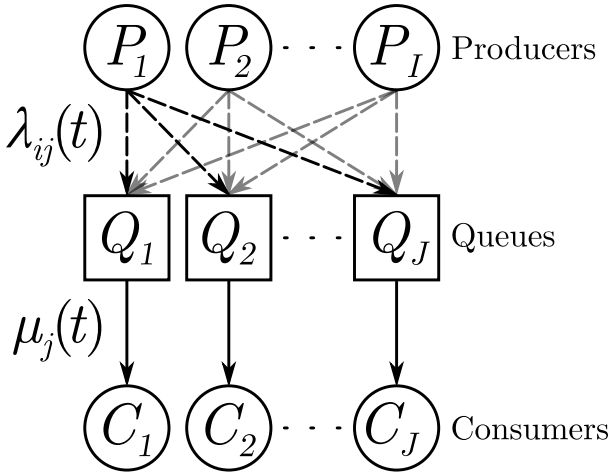


Figure 3.1: An abstract representation of the job pipeline

Figure 3.1 shows an abstract representation of the job pipeline in this context. Jobs start at one of I producers and are distributed to one of J queues that are serviced by as many consumers.

Messages arrive at queues from each producer P_i in a stochastic process. This process is a superposition of $\lambda_{ij}(t)\ \forall j \in [1, J]$ at any given time slot t, where $\lambda_{ij}(t)$ denotes the message arrival rate from a producer P_i to a queue Q_j at time slot t and $\mu_j(t)$ the service rate for consumer and queue C_j and Q_j at time t.

Each consumer queue can potentially be on a different host. Querying metadata, such as queue length, can therefore take time that is on the same order of magnitude as sending a job to a queue. This means that decisions that take this metadata into account, either must incur a significant delay, or operate on stale information.

Given the context, we are constrained by a number of conditions:

- Heterogeneous consumers
- Unknown, time-dependent number of consumers
- Unknown, time-dependent arrival rates
- Unknown, time-dependent service rates
- A producer's view of consumer metadata is not instantly updated

Each producer must decide which queue, and this which consumer, to send a job to. Each consumer-job pair will have a different job completion time. This directly affects queue-lengths, which also affect job completion time. An ideal solution must therefore take account of this. As discussed in § 2, most load balancing methods do not take this into account primarily as they assume homogeneous consumers.

Jobs requires access to certain data sources. Certain consumers might be riskier to grant access to this data to based on where they are, who they are controlled by, and what other data they have had access to in the past. The jobs must be distributed in such a way that privacy is considered.

3.1.1 Performance metrics

Our measure for performance is the mean time it takes for a job to complete from arrival into the system to result response. We refer to this as mean job completion time. As a metric to compare solutions against each other with, this is just as valid as sum job completion time except more intuitively comprehensible.

We also consider the variance in job completion time indirectly as this captures the unwanted cases where most jobs might complete relatively quickly with a few unacceptably slow outliers.

3.1.2 Related work

Our problem space exists at an intersection of job scheduling and load balancing. The body of existing research in scheduling, queuing theory, and load balancing is extensive. This section describes the most recent and pertinent work in these areas.

Load Balancing

In the area of load balancing[66], the hash-based or random load balancing such as Equal Cost MultiPath (ECMP) has been widely deployed in modern datacenters [52, 32]. These load balancing schemes fall short on asymmetric load balancing due to potential topology or resource unavailability, and they do not take account of congestion well. Similarly, other schemes such as Presto [50] operate on near-uniform units of data (i.e., "flowcells") in a round robin fashion, and rely on other optimizations. In our context, this is not practical as asymmetry and imbalanced congestion are inherent.

Some recent load balancing schemes that take account of traffic conditions, such as CONGA [2] and HULA [65], are distributed and rely on switches to track congestion. In our context of edge computing however, the cost of querying traffic conditions is significant and the load balancing must be done with stale information. Common load balancing algorithms have been organized into taxonomies in the past [66].

Honey bee foraging [77], while suitable for dynamic environments, cannot be validly compared to our proposed method as it applies to distributed load balancing. Load Balance Min-Min (LBMM) [72] and its max-min sibling [10] consider job execution time when scheduling. However, this time needs to be known in advance and homogeneity between nodes is assumed, both of which are not always possible in our context.

Job Scheduling

Queuing Theory

When applying queuing theory to our problem, one common assumption is that servers (edge devices in our context) are identical and have a stationary service rate (i.e., how fast a job can be serviced or processed) distribution [123, 53, 89, 82]. Unfortunately, we cannot just model service rate as a stationary process even in short time windows, as the service rate is subject to sudden changes caused by, for instance, additional load from new data sources, and new devices entering or leaving the system. Similarly, online optimization approaches such as backpressure routing [97] only take account of queue

lengths, and thus assume identical service rates for all the queues.

In another thread of research, approaches that consider non-identical queues and describe optimal queuing decisions to minimize job response time are often referred to as Shortest Expected Delay Routing (SEDR) or Shortest Delay Routing (SDR) [54, 74, 81, 60, 108].

These approaches, however, still model service rates as stationary processes, which is not true in our context. If service time is stationary, it is trivial to estimate, and the problem reduces to a simple case of load-balancing based on expected wait time. This is however not the case in our context.

Online optimization approaches, mainly in the ad-hoc wireless network space, make routing decisions based on throughput and latency [45, 125, 14, 95]. This kind of congestion-aware routing assumes limited knowledge, measures service rates to adjust routing decisions, and combines shortest path routing with back-pressure routing to optimize for congestion and latency. Unfortunately, these approaches do not consider the fact that the delays in querying queue-related information are significant, and thus, job distribution must be done using the stale information.

This chapter focuses on the second area that we explore privacy-preservation in as foreshadowed in § 1.3. We begin by contextualising job distribution and the privacy risks involved, then we illustrate our design for augmenting such systems with privacy-preservation with no net cost to performance, and finally we evaluate our solution in terms of privacy, performance, and the trade-off between the two.

The solutions we present in this chapter are generalisable and applicable to any system where arbitrary jobs need to be distributed over arbitrary consumers. However, we take advantage of certain characteristics of the context to which we deploy, to optimise performance and solve additional constraints.

3.2 Optimal edge job distribution

We have established how an important property of edge networks is the fact that they are heterogeneous. In this section we present our solution for optimal distribution in this context that takes advantage of this property to outperform more general purpose, context-agnostic solutions.

3.2.1 Design

This section describes our system and the justifications for our design decisions. Throughout this section, we will use the terminology of a message broker system (e.g., Apache Kafka) to tie the design to our implementation in §3.2.2.

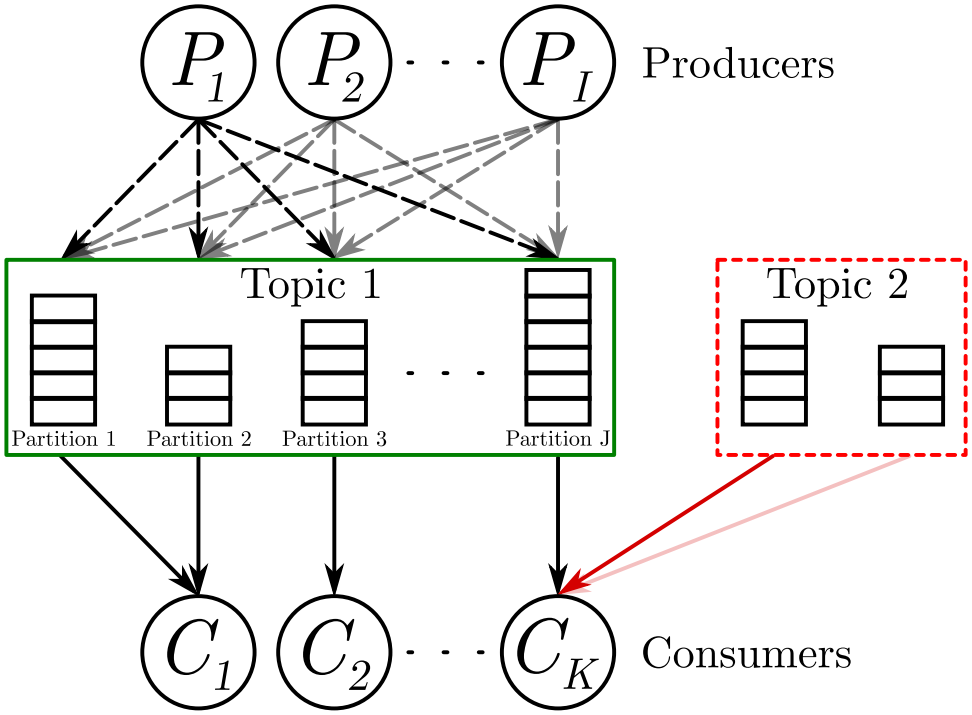


Figure 3.2: An overview of a message broker system similar to Apache Kafka. I producers publish messages to J partitions which are serviced by K non-homogeneous consumers.

Background and Terminology

The architecture of existing message broker systems is fairly standard. Message brokers may be centralized, or distributed over a number of nodes for scalability and resilience through replication. Messages are mapped by *topics* (see Figure 3.2); e.g., all messages from/to an application/service can form a topic. A topic can be split into multiple disjoint *partitions*, each of which can be treated as a *queue* and functions as a unit of concurrency.

In addition, there are *producers* that put messages into the message broker system, and *consumers* that read messages from the system. More specifically, a producer can publish messages into multiple partitions, but any partition can only be subscribed by (at most) one consumer. These *shards* (e.g. in AWS Kinesis) or *partitions* (e.g. in Apache Kafka) can be treated as queues, and function as a unit of concurrency. In Kafka, messages are split into *topics* that consumers can individually subscribe to.

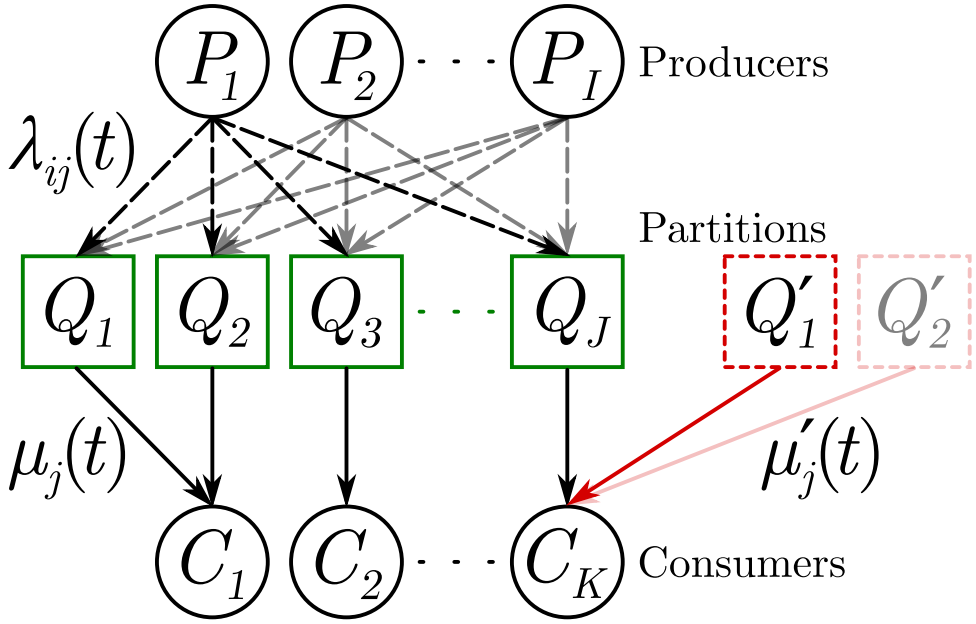


Figure 3.3: A generalized view of a message broker system. I producers publish to J partitions which are serviced by K non-homogeneous consumers.

Figure 3.3 shows how this architecture maps to a general queue case. It depicts the paths that messages take, starting at one of I producers, passing through one of J partitions in a topic that behave as queues, to finally be consumed by one of K consumers. Messages arrive from each producer P_i in a stochastic process. This process is a superposition of $\lambda_{ij}(t)$ $\forall j \in [1, J]$ at any given time slot t, where $\lambda_{ij}(t)$ denotes the message arrival rate from a producer P_i to a partition Q_j at time slot t.

Producer arrivals, with an arrival rate of $\lambda_i(t)$, can be thinned/split in such a way that the proportions of $\sum_{i=1}^{I} \lambda_{ij}(t)$ are identical to $\mu_j(t)$ across queues, where $\mu_j(t)$ is the service rate for Q_j. On the other hand, each partition Q_j is consumed by one consumer with the instantaneous service rate $\mu_j(t)$, i.e., how fast the messages in the partition Q_j are serviced or consumed by the consumer at time slot t. The assignment of partitions to consumers can be rebalanced periodically as consumers enter or leave the system, which would affect the service time of each partition.

Problem Statement

In this chapter, we consider a scenario where the job/message distribution runs on the network edge. There are a number of partitions, each with a varying and non-stationary service rate distribution. A producer must decide which partition to publish a message to, based only on the stale information of partition lengths. The problem we seek to solve is to minimize the job response time (defined as the service time plus the queuing delay) of

messages, and thus the completion time of jobs. What makes this problem non-trivial is that, in the edge computing context, we further have the constraints of non-homogeneity that is not static, as well as a stale view of queue lengths. A practical solution must take account of the following characteristics:

- Producers can only use partition lengths to make message distribution decisions.
- The time to query these partition lengths is significant and costly, thus necessitating the use of stale information.
- Partitions may have volatile service rates due to:
  - Various edge device capabilities and resources,
  - Imbalanced assignments when the number of partitions is not a multiple of the number of consumers (see C_1 in Figure 3.3),
  - Additional load from the subscriptions to partitions of other topics (see Q' in Figure 3.3),
  - Extrinsic processes running on the same hosts as the consumers.

Our solution combines standard queuing decision methods, with convolutions and heuristics for estimating service rates, and load-balancing strategies that take account of stale information.

Assumption and Overview

We assume that a message in a topic requires the same or a similar amount of *work* as all other messages in that topic (regardless of which partitions in the topic). However, the capabilities and resources of consumers differ across devices, and therefore, the time it takes to service a message from a topic may differ across consumers.

Service rates are therefore per-partition; they can vary based on sudden stimulus such as new partition assignments to a topic, or extrinsic processes starting on the same device as a consumer. Figure 3.3 demonstrates this issue with imbalanced partition assignments and topic subscriptions. Assuming saturated partitions and identical consumer capabilities, service rates $\mu_1(t)$ and $\mu_2(t)$ will be smaller than $\mu_3(t)$ due to the two associated partitions Q_1 and Q_2 both being serviced by the same consumer C_1. Meanwhile, the consumer C_K additionally subscribes to partitions Q'_1 and Q'_2 of a different topic (red, dashed line), which effectively add extraneous load on the consumer C_K, affecting how quickly it services messages from the partition Q_J of the other topic (green, solid line).

In cases where these factors are slow to change, a simple moving average with a constant window size is good enough to estimate the accurate service rate of a partition. In other cases, more sophisticated methods are needed, such as a Kalman filter [62], or even machine learning approaches to additionally take account of seasonality such as daily patterns.

With an estimate of the service rate $\hat{\mu}_j(t)$ of each partition Q_j, we adjust the probabilities that govern how the arriving messages are partitioned. To do this, we use the service rate estimates in conjunction with the partition lengths to compute the per-partition *expected response times*. Since the information of partition lengths may be updated at a slower rate than message arrivals, we operate with cached partition lengths. In particular, we employ the "best of two" partitioning strategy [106], i.e., first select two random partitions and then choose the one which has a lower expected response time. This makes our partitioning strategy near-optimal[1]. This "best of two" strategy has been proven to be the most ideal partitioning strategy with stale information [106].

Our Approach

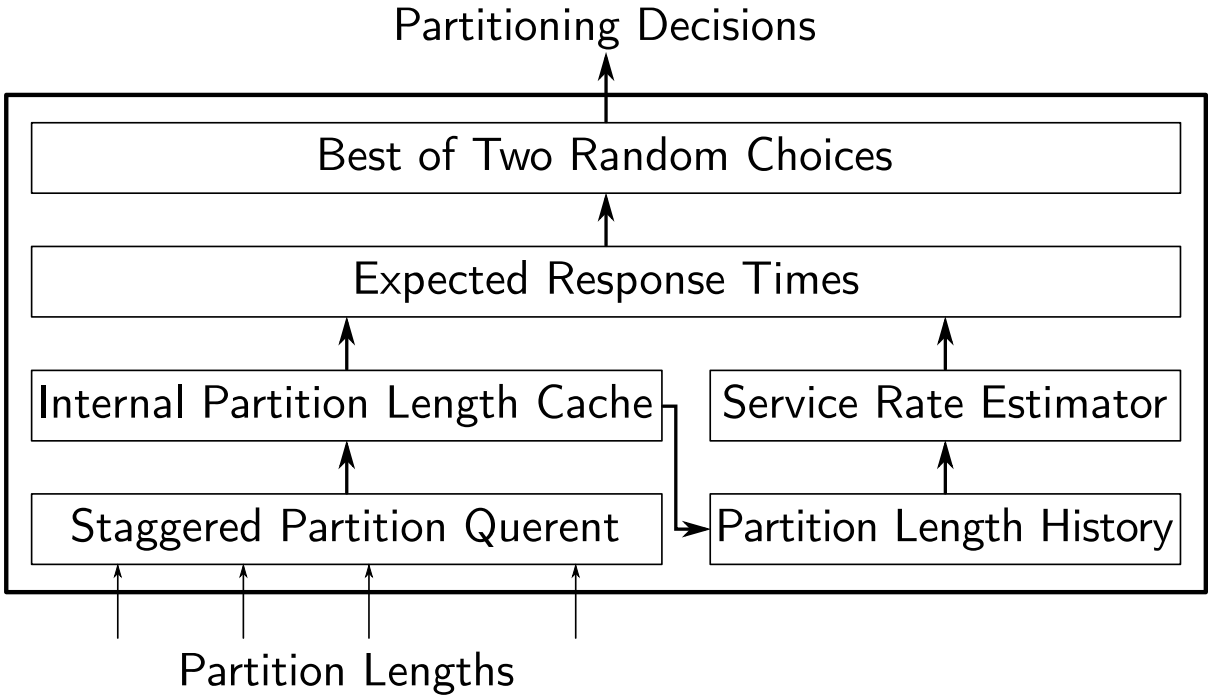


Figure 3.4: Producer components and steps, for generating the partitioning decisions from stale partition lengths.

Figure 3.4 illustrates how each producer determines which partition to publish a

[1]Note that the optimum assumes that the actual service rates are known and the cache refresh interval is 0.

message to. The *staggered partition querent* queries and updates the stale partition lengths in a producer's *internal partition length cache*. The querent is independent of the process that does the partitioning, for instance, via running in a separate process or thread. In order to avoid having producers query partitions for queue lengths all at once, each producer distributes its queries uniformly over the cache refresh interval, which is equivalent to the minimum query delay (roughly equal to the round-trip time between the producer and partition nodes). The producers can further offset these query times such that each producer is out of phase with every other producer. This way, different producers additionally do not query the same partition at the same time, which further minimizes any delays caused by congestion.

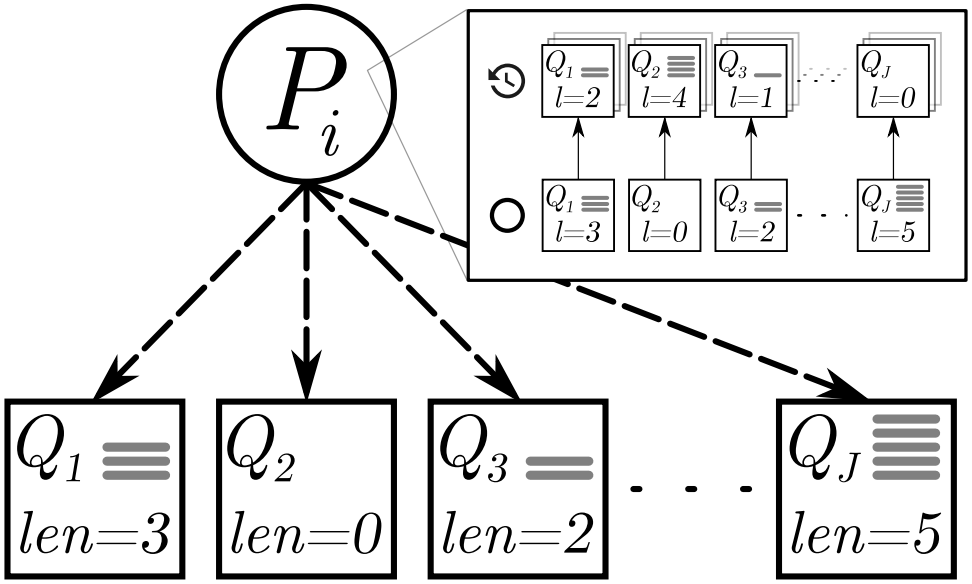


Figure 3.5: The internal components of a producer. Each producer periodically queries partition lengths and stores a history of past partition lengths.

Each producer maintains a *partition length history*, as shown in Figure 3.4 with more details in Figure 3.5. The producer then uses this partition length history to *estimate the service rate* currently expected for a particular partition. As previously mentioned in §3.1, this estimation can be done in a variety of ways depending on the use case.

The *best of two random choices* strategy is applied on top of the *expected response times* which are computed via dividing (cached partition length + 1) by the estimated service rate, as depicted in Figure 3.4. For every message arrival, a producer selects two random partitions anew, and sends the message to the partition with a shorter expected response time.

Queuing decision based on cached partition lengths is a heavily-studied problem [106]. When partition lengths are cached/stale, simply selecting the partition with the shortest

expected response time will create oscillations. This is because all producers crowd into the best partition, thus creating a significant imbalance before the caches get updated, after which they all crowd into a new best. It has been proven that selecting the best of two random partitions is the ideal strategy that does not require coordination [106]. It is therefore the strategy we employ in our system.

Discussion

The advantage of our system over existing approaches is that, by postulating that producers can only query partition lengths periodically, distributed systems that use our method will scale better, because the consumer assignments do not need to be known. Similarly, by having each producer maintain an independent cached partition length history, no coordination between nodes is required and no central component is needed to do so, thus avoiding a single point of failure. For many practical applications, the trade-off between decentralization versus repeating queries and storing duplicate data weighs largely in the favor of decentralization, as the number of partitions is practically on the same order of magnitude as the number of producers.

Additionally, our system can be adjusted to become more or less aggressive, depending on the volatility of service rates. Similarly, the convolutions for estimating service rates from past information can also be adapted in different contexts.

3.2.2 Implementation

This section describes our system implementation. Our approach assumes a job scheduler that uses a message distribution mechanism to assign messages to edge devices. Although our approach is generalizable to any message distribution mechanism, we describe our implementation based on Apache Kafka as it is already used for the purpose of distributed computing in the cloud in the form of *serverless computing*, and serverless computing has been extended to distribute computation on the edge in the past [41].

Figure 3.2 also illustrates how our use case maps to Kafka's queuing model: A consumer subscribed to a topic pulls messages from the partitions that are assigned to the consumer by the system. When a group of consumers is subscribed to the same topic, the system will attempt to balance partition assignments, such that each consumer pulls from as close to the same number of partitions as possible. A partition can only be assigned to (at most) one consumer.

In Kafka, a message published to a particular topic is routed to a partition based

on the producer's strategy via a *partitioner*. We exploit this ability to query partition lengths, and use these partition lengths along with estimated service rates to compute the per-partition expected response times (see §3.2.1). Our partitioning decision uses cached partition lengths, because querying the latest partition lengths for each new message incurs significant overhead.

We have implemented our system as a standalone `class` that implements Kafka's `Partitioner` interface. The custom partitioner can be used by setting the appropriate configuration property for the producer. Our implementation queries partition lengths and performs the best-of-two random choice strategy in separate threads. Until the partition length cache is populated during the initialization, our partitioner resorts to randomized partitioning.

3.2.3 Evaluation

In this section, we first present the simulation results to demonstrate the feasibility of our approach. We then present some results based on our real system implementation and show that the overhead of our approach is virtually not worse than that of the partitioning strategy used in Apache Kafka.

Simulation

To demonstrate the feasibility of our approach, we simulate a set of partitioning strategies. We compare the performance of our approach with common partitioning strategies that are present in Apache Kafka: hash-based partitioning based on the key of a message, and round-robin partitioning. These two strategies work independently of service rates and partition lengths.

In addition, we repeat our simulations with a naïve "least-load" strategy, where only partition lengths are considered and service rates are ignored (i.e., Lyapunov optimization). For this strategy, we also simulate different querying intervals of 1, 10 and 100 ticks. Here, a tick is a unit of time for simulation, and only the relationship between the arrival and service times matters.

We simulate a scenario in which two producers produce messages as a Poisson process, with a mean arrival time of 100 ticks, into a topic that has three partitions. Two consumers pull from these partitions as a Gaussian process with a mean service time of 100 ticks and a standard deviation of 50 ticks. These parameters allow us to overload the network in our simulation, so that we do not report trivial message response times

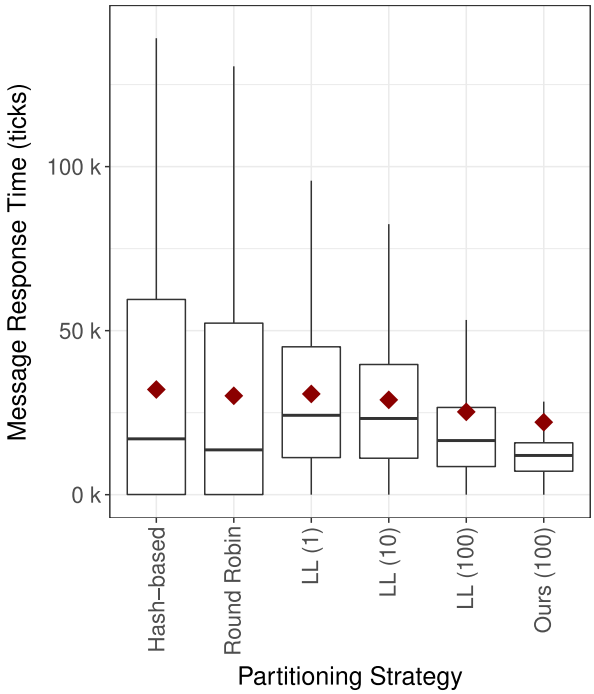


Figure 3.6: Message response times with different partitioning strategies as Tukey Box-Whisker plots. Whiskers show 1.5× inter-quartile range above (respectively below) the third (respectively first) quartile, whereas red diamonds show the mean values.

without load.

Choosing three partitions with two consumers results in an imbalance of partition assignment with one consumer being assigned to two partitions, which allows us to simulate heterogeneity. Additionally, we add other topic subscriptions to one consumer, so that its capacity and its service time for its partition are both negatively affected.

Figure 3.6 shows the distribution of message response times for each evaluated message partitioning strategy. As shown in Figure 3.6, decreasing the querying interval of partition lengths causes the response time of messages more unpredictable for the "least-load" partitioning strategy (i.e., LL(x) with x being querying interval). Our method uses a querying interval of 100 ticks to obtain partition lengths, and makes estimation of future service rates as well as the expected response times, based on only the cached values of partition lengths.

As Figure 3.6 shows, our method outperforms all other partitioning strategies in mean (red diamond) and variance values. Particularly, the mean message response time is reduced by 31.08% and its variance by 51.71% when compared to Kafka's default hash-based partitioning strategy.

We conclude that, when the partitioning decision is made based on stale and rapidly

fluctuating information, our system results in a more optimal message distribution with both lower mean and variance, and therefore faster and more predictable overall performance.

Real System

To evaluate the practical feasibility of our method in a real message distribution system, we generate load and measure the time for message partitioning in both vanilla Apache Kafka and our implemented system. In particular, we compare our partitioning strategy with the default Kafka partitioning strategy by streaming a total of 100,000 messages. Our producer and consumer setup is the same as in our simulation, i.e., a topic with three partitions and two consumers in an imbalanced partition assignment.

Our implementation uses multiple threads, so that it can utilize multiple cores. We offload the querying of partition lengths to a separate thread. This approach enables our partitioner's performance to be on the same order of magnitude as Kafka's default hash-based partitioner.

As such, the mean execution time of our partitioning function is 957 ns with a median of 655 ns, whereas Kafka's default partitioner has a mean execution time of 482 ns and a median of 368 ns. The difference between mean and median for our implementation is due to the bimodality in the distribution that can be explained by greater churn with resource allocation, and therefore, garbage collection phases.

Although our system adds some overhead in making partitioning decisions, we find this overhead negligible because the reduced message response times (as shown with the simulation in §3.2.3) substantially outweighs this overhead. As a result, our partitioner suits well for the edge computing scenario which requires low latency.

3.2.4 Summary and discussion

In this work, we motivated the need for a dynamic job distribution system that optimizes for latency in edge computing. In such a scenario, edge devices executing messages are spatially and temporally non-homogeneous, making it challenging to find message assignments that reduce the message response time, and thus the job completion time.

We presented the design and implementation of our approach that exploits cached partition/queue lengths to predict the service rates and the expected message response times. Combining this information with an effective heuristic to pick the best-of-two random partitions enables our system to outperform other common partitioning approaches

to reduce response times by 31.08% with virtually no extra overhead. As a result, our approach significantly helps lower job execution times in edge computing scenarios.

hat takes advantage of available information in Apache Kafka (lags) and extends Kafka partitioners. Our paritioner makes use of these lags to predict consumer service times based on past history and selects the best of two random choices.

We evaluated our system in a simulated environment and compare it to other approaches. We have shown that in this context, our system has the propensity to out perform others and achieve lower job execution time thus providing better user experiences in domains such as IoT while also saving power, money, and computation.

Future Work

Future work includes the investigations to optimize not only for latency, but also for resource usage, cost and privacy of edge devices. We intend to compare these metrics, when using different message distribution strategies, to understand the trade-offs between employing more nodes to handle increased workload and balancing the workload between existing nodes.

Additionally, we plan to compare our approach with more practical message distribution strategies, as well as use our Kafka-based implementation to evaluate more real-world use cases, including these recent serverless computing paradigms [51, 76].

In order to more rigorously evaluate our method and iterate on it, we intend to evaluate our Kafka partitioner in real use cases such as in the paradigm of serverless computing.

We also intend to investigate optimising for other factors (as opposed to latency) such as resource usage, cost, and even privacy context.

We intend to compare other performance metrics between strategies, such as overhead in resources, and trade-offs between better load balancing versus more nodes. As this is a currently unexplored area, the potential for improvement is significant.

3.3 Augmenting edge job distribution with privacy-awareness

In this chapter so far, we have explored the distribution decision with the sole goal of minimising job completion time. As previously alluded to however, this is not the only

factor we need to take into consideration. This section explores how we can build in a notion of privacy awareness to augment the above system and similar distribution systems to take privacy into account. We first build and evaluate a model for objectively measuring privacy, then design, implement, and evaluate a privacy-aware job distribution system.

3.3.1 A framework for defining "privacy"

When it comes to personal data, traditional access control mechanisms between a *producer* and a *consumer* of data exhibit critical problems. With social media APIs, phone sensors, and even files on a PC, access is most often a binary "all or nothing". While in some cases access can be attenuated to read or write, and can expire after a certain time period, this level of granularity is too coarse, and does not consider the content of the data in any way.

APIs that do allow more fine-grained controls often require an understanding of the context and possible inferences [17], to then allow a user take context-specific actions such as spoofing GPS coordinates or occluding faces in images. While this is useful, it is difficult to scale and generalize to arbitrary data types without user interaction or complexity for understanding semantics.

Furthermore, users are often unaware of just what information they really are exposing [3], and cannot be expected to keep track of inferences that may be caused by anomalies or patterns in their data, especially not in real time. Dynamically adjusting access control restrictions based on online privacy and risk metrics remains an open problem.

These shortcomings culminate in access control mechanisms with only very superficial privacy awareness. The goal of this work is to introduce a scalable, privacy-aware access control system that solves these problems.

We seek to do this by applying established, information-theoretic privacy metrics as criteria in access control systems over time series data. These metrics must be *context-independent*, operate in *real time* on *cheap hardware* located *at the source* of data. These constraints are imposed by the context in which we deploy this system and evaluate its performance and extent of privacy preservation.

While our method is applicable to any system where a consumer pulls personal data from a producer under the restrictions of an access control system, we implement and evaluate it in the context of the Databox platform [18] — a home-based networked device

that provides a controlled, sandboxed environment for processing personal data. We discuss this evaluation in greater detail in the next chapter, while in this chapter we focus on the privacy aspects and how they can be combined with job distribution.

In this section, we describe an implementation of this using information theoretic privacy metrics on time series data. We then evaluate our system's privacy-utility trade-off, as well as its performance in real-time, low-latency use cases, such as in embedded and home IoT devices. We show that our system provides privacy gains without a significant utility cost, and can run efficiently and scale well on cheap hardware with high-frequency sensor data.

Our system

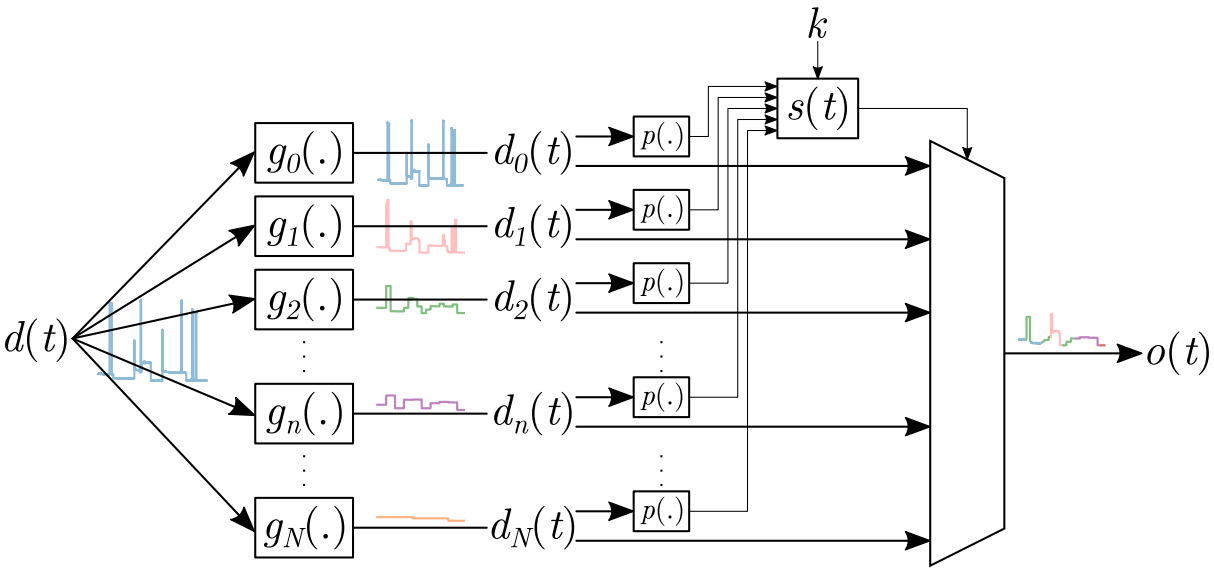


Figure 3.7: An overview of of our implementation

This section details our implementation and describes figure 3.7 which provides a visual overview of it.

We begin by transforming time series data $d(t)$ continuously to N different granularities. This can be for any definition of "granularity", however in our implementation, we calculate the mean of every 2^n samples for $n = 0, \cdots, N$ and interpolate by nearest neighbour. Alternatives include plain downsampling, summing/aggregation, or other forms of averaging. We calculate means as these have the greatest utility to our evaluation use case.

We denote these granularity transformation functions as $g_n(x)$, the outputs of which map to the original time series in the following manner: $d_n(t) = g_n(d(t))$.

For every new sample in the transformed data, $d_n(t)$, we update one or more corresponding privacy scores based on the privacy metrics described earlier. For our evaluation, we use surprisal so the unit of these scores is bits or shannons. We describe how we implement this and other privacy metrics in more detail in the next sections. We denote this privacy measure as the function $p(x)$.

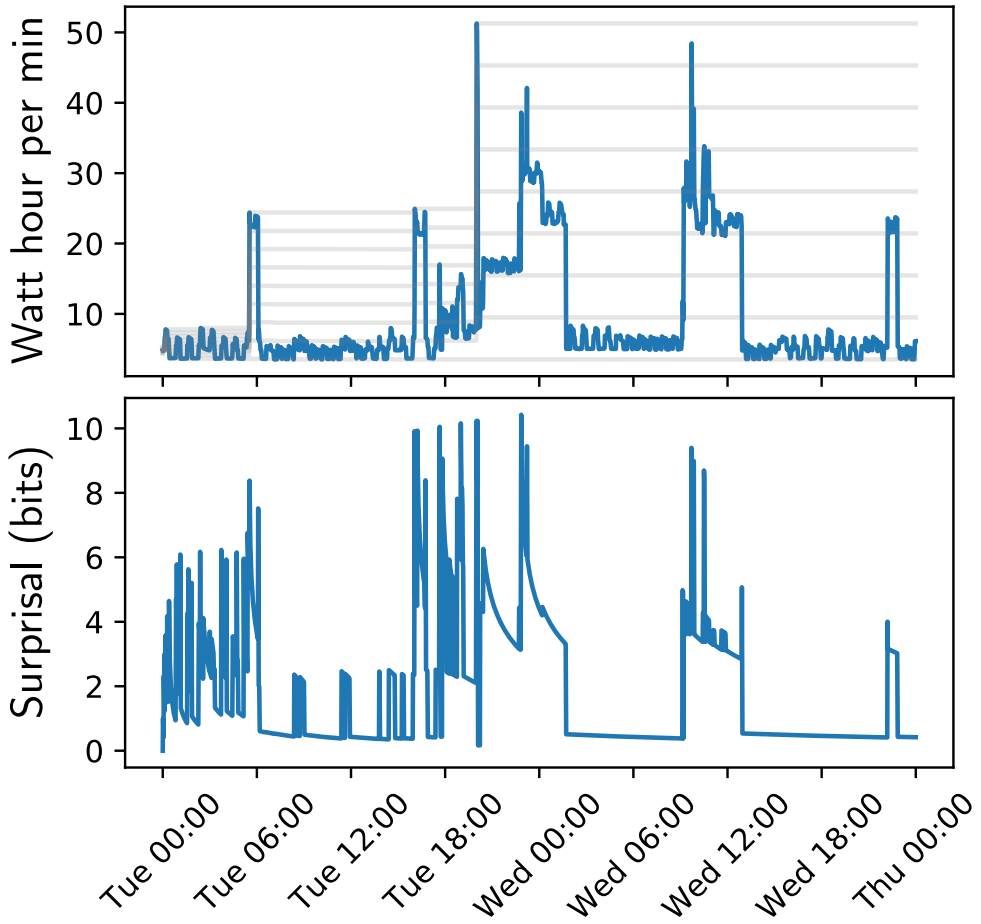


Figure 3.8: Surprisal over active energy consumed each minute with eight bins (gray) and an infinite window size

The final component in this system is a multiplexer that selects the transformed data stream with the highest granularity but with a privacy score that is still below a threshold k. The moment a data stream's score exceeds k, the multiplexer drops to a lower granularity, until it reaches the level of granularity that is the equivalent of a fixed grand mean across the entire stream. This is the point at which $n = N$. With our previously defined notation, our final output is $o(t) = d_m(t)$ where $m = \min\{n \mid n \in \mathbb{Z} \wedge n \in [0, N] \wedge p(d_n(t)) < k\}$. In other words, the highest resolution transformed stream with a privacy score below a given threshold.

$$d_n(t) = g_n(d(t)) \tag{3.1}$$

$$s(t) = \arg\max_{\substack{n\in\mathbb{Z} \\ n\in[0,N] \\ p(d_n(t))<k}} d_n(t) \tag{3.2}$$

In equation 3.2, $s(t)$ is a shorthand to represent a function that returns the index of the data stream selected at time t used in figure 3.7. For a sample $d(t)$ at time t, the overall output can be formulated as a dynamic optimization problem. Here, selected data stream index n is our decision variable.

$$\begin{aligned} \min_{n} \quad & d_n(t) \\ \text{s.t.} \quad & n \in \mathbb{Z} \\ & n \in [0, N] \\ & p(d_n(t)) < k \end{aligned} \tag{3.3}$$

The threshold k is embedded in the tokens attached to requests made by data consumers. By modifying k for a consumer, we can modify the extent in permissions with respect to the metric used, and thus achieve privacy-aware access control.

It is important to note that dropping to a lower level of granularity is done *transparently* without the consumer's knowledge. We also test a variant of this system where the consumer directly requests a specific level of granularity, and their request is rejected if it is above what they have permission to access. The consumer must then make a new request to a lower granularity, which not only adds significant latency, but the act of denying access itself reveals some information on the nature of changes in the data, for instance if a stream that a consumer originally had access to suddenly became restricted.

As a proof of concept, we run the metrics described in the previous section over UCI's Individual Household Electric Power Consumption Data Set [80]. We focus on just two columns from this dataset: the timestamp column, and the global (minute averaged) active power, which we convert to watt hour. In order to more closely conform to realistic scenarios, we treat the range of this data as an unknown that is continuously updated as the maximum and minimum seen values are exceeded.

As data processed is continuous, it must be quantized first for certain metrics in order to become discrete. This is not the case for input data that is for example a byte of sensor data in the range $[0, 255]$, or strings from a set of limited size like country names.

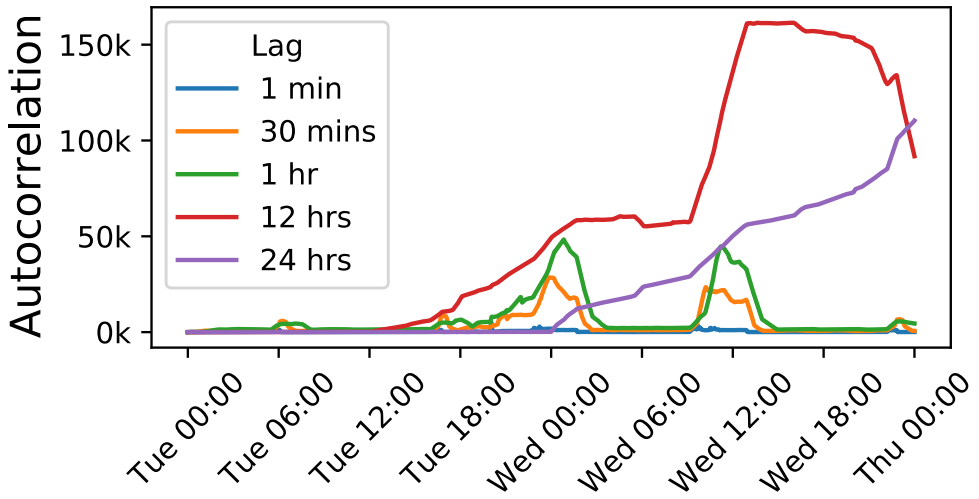


Figure 3.9: Autocorrelation over power consumption for different fixed lags

With continuous variables however, the probability of any one sample is near-zero and so we must divide the data into bins of a set interval in order for the metrics to make sense.

Furthermore, as we perform these measurements online, it is practically infeasible to repeat these for the entire dataset on every new addition. In our final implementation, we therefore only consider the data before a temporal cutoff point using simple, fixed-length, rectangular windows. We note however that windows with different configurations and weightings may potentially yield cleaner results.

In figure 3.8, it is clear that as samples are added to empty bins, surprisal spikes. As more are added to one bin (such as around Wednesday at midnight), surprisal slowly decreases, which makes sense intuitively. Suprisal is lowest for power consumption values below 10 watt hour, as this is the most replete bin.

To measure seasonality, we track autocorrelation across a set of lags over time. This is different than simply building a correlogram over a fixed length sequence, as we continuously calculate cross-correlation in an online manner. Figure 3.9 visualises the most significant lags. For 12 hours, autocorrelation is mostly higher than for 24 hours, which would also hinted by maxima in a correlogram. This tells us that patterns are most likely to arise at 12 hour periods, and we can automatically take steps to suppress this by for instance only emitting 12 hour averages. We can also normalise this data and embed cross-lag, or per-lag autocorrelation thresholds into our access control system. This could for example block access to data with exceedingly high 12 hour autocorrelation, but allow access to the same data only once it has been downsampled to obfuscate this pattern. As such, this is a simple, yet powerful metric to use in our system.

The privacy-utility tradeoff

Our system provides privacy along the axis of the privacy metric used based on the thresholds used. For example, tokens that only allow access to data with a surprisal value of less than four bits will implicitly favour less granular data. An emergent effect of this property is that higher frequency inferences can be suppressed in aggregate time-series datasets.

To test this, we take the same approach as previous work in the same domain [73] and make use of the Reference Energy Disaggregation Data Set (REDD). This dataset contains detailed power usage data from a number of houses including mains readings as well as on a per-device basis.

Our goal is to show that after subjecting aggregate mains data to our privacy-aware access control system, we can infer washer/dryer state while concealing microwave state. A realistic use case may be a smart meter app that suggests the best times to do your laundry based on your flatmates' habits, but does not need to know anything about your eating habits. Our system allows for this gain in privacy without compromising utility.

We use the state (on or off) of the washer/dryers and microwaves as ground truth and a Gaussian Naive Bayes classifier to predict whether or not these devices are turned on given mains data. The data has occurrences of both devices being turned on both separately and together. For each device, we plot Receiver Operating Characteristic (ROC) curves where we modify the privacy metric threshold, which is in this case an upper bound on bits of entropy.

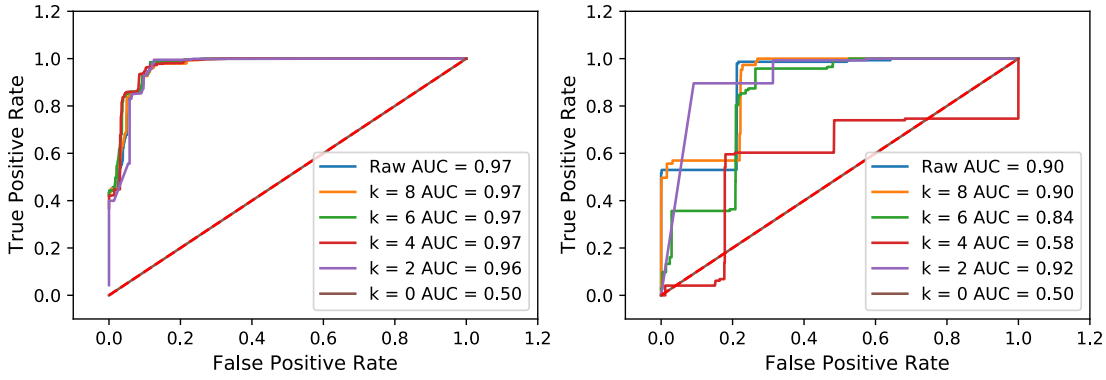


Figure 3.10: Receiver Operating Characteristic (ROC) curves for washer-dryer (utility; left) and microwave (attack; right)

In reality, an app might not have access to household-specific data to train such classifiers. We show however that even in this case (figure 3.10) utility remains virtually the same across thresholds while the undesired inference degrades.

3.3.2 Privacy-aware job distribution

As we have established in § 1.2, there is a trade-off between privacy and performance. This trade-off usually manifests itself as a time penalty for achieving the same level of utility with transformed data, or added overhead in how the data is processed.

In § 4 we have shown that it is both possible and feasible to enforce restrictions on a system of data producers and consumers in such a way that privacy is observed. While this approach is versatile, it does not consider the wider picture: why should a risky consumer want to query sensitive data in the first place?

This leads us to a more fundamental question on what decides which consumer is tasked with processing certain data. In § 4 we have shown that there is value in performing computations on edge devices in terms of both privacy and performance. The parameters that affect these, and to what extent they do, is unexplored. We aim to explore this through simulation and performing a sensitivity analysis on these parameters.

We intend to take this one step further and claim that for any series of jobs, there exists an optimal way to distribute these such that we can achieve an optimal trade-off between privacy and performance (for the same level of utility).

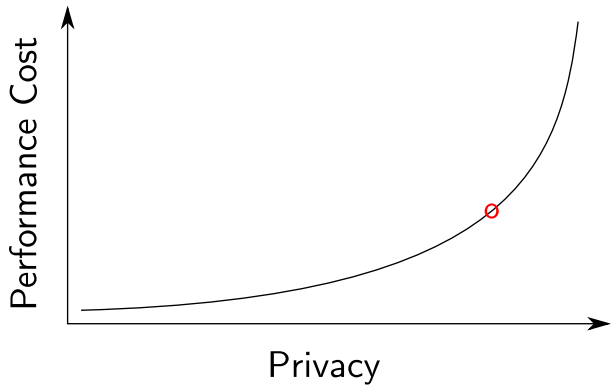


Figure 3.11: An abstract representation of the optimal trade-off point between performance and privacy

In figure 3.11 we simplify this trade-off to explain how understanding this it in a general way can allow both the user and the system to make policy decisions that will balance performance with privacy.

With naive distribution strategies, such as a hash-based or round-robin strategy, this curve is normally linear. This is because trusted consumers introduce performance penalties, and these increase at a constant rate the more trusted consumers you add.

However, the curve will have the above shape in cases where the job distribution method considers performance only. Strategies that do this try to utilise the faster (albeit

less secure) consumers fully before falling back to the trusted ones. The more the ratio shifts in favour of trusted, the less choice these have, and we see a steep increase, giving the curve its "knee".

Privacy measures can be tuned to the point of diminishing returns — where the curvature of the curve is at its maximum, a point which can be found computationally.

Furthermore, we aim to develop intelligent load balancing techniques that exploit characteristics of this deployment context to improve performance overall. We can then show that using these techniques can allow the practical implementation to achieve privacy at little to no performance cost.

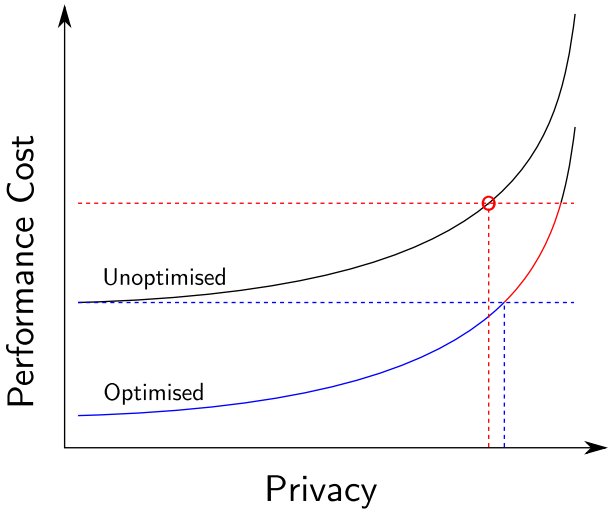


Figure 3.12: An abstract representation of the possible levels of privacy for an optimised implementation that would perform better than an unoptimised one

Figure 3.12 aims to depict this; our optimisations would push the performance cost down overall, allowing us to enforce privacy at up to a point where performance would be better than or equal to an unoptimised implementation with no privacy measures. This is the blue segment of the line. As a bonus, this level of privacy can also be higher than that of the earlier red point.

To demonstrate the practicality of our solution, we first perform a sensitivity analysis over the parameters involved, to determine how these parameters should be set in a practical implementation and which have the greatest effect on the outcome. Then, we use these results to set up testbed for comparing distribution strategies, to show where our strategy is better and inform us on how we can improve it.

Sensitivity analysis

We simulate our methods adding privacy-awareness to job distribution systems for two main reasons.

1. To check if the outcomes match what is expected and that our assumptions are valid.
2. To perform a sensitivity analysis of the different parameters and determine which have the greatest effect on our results.

Setup and Methodology

We simulate a testbed of 30 consumers. Some of these consumers are *normal* and some *private*. TODO: refactor private to trusted To simulate private consumers, we draw on experimental overhead measurements for trusted computation in SGX. [49]

The simulation creates jobs with specified units of work needed for completion. These jobs arrive into the system dictated by a specified process and process parameters. On arrival, the system distributes these jobs to a consumer queue based on a given job distribution strategy. The (heterogeneous) consumers then service these jobs until the work, and therefore the job, is complete.

The simulation is set up in such a way that when the majority of the consumers are private, the arrival rate of jobs outweighs the service rate. This is intentional, as it illustrates what happens when the system is in a top-heavy state.

All plots are distributions of results over 100 simulation repetitions. We have found that more than that has no significant effect on the distributions.

Table 3.1: Simulation parameters, bounds, and description

Parameter	**Bounds**	**Description**
Number of consumers	$[1, 30]$ why	
Subset of private consumers	$[1, 30]$	
Distribution strategy	*See below*	
Number of jobs	20000 why	
Job arrival process	$\in \{discrete, poisson\}$	
Job arrival rate	$[1, 3]$ why — top-heavy	
Job work units	45 why, job types?	
Queue length local cache refresh interval	$\in \{0, 10, 100\}$ why	

Table 3.1 shows a list of parameters that our simulation takes as inputs. Our sensitivity analysis is done within the bounds of these parameters.

We also compare other common distribution strategies as baselines and controls. These are as follows.

- **First** — The consumer with the lowest index is always selected. This is "bad" load balancing.
- **Round-robin** — A common naive distribution strategy that works best when consumers are homogeneous.
- **Hash** — Hash-based distribution is a very common way to have consistent grouping of jobs and consumers as the decision is based on a hash of job keys.
- **LeastLoad** — The consumer with the least load (smallest queue) is selected. This strategy works best when jobs are homogeneous.
- **Our** — Our optimised method, which we show is better than the other common methods in this context.

In order to reduce the parameter space, we also made certain assumptions and simplifications. These are justified in the following section.

Assumptions and Constraints

In our context, we make a number of simplifying assumptions this section, we enumerate and justify the assumptions made and constraints imposed by simulating this system in our context.

1. **Poisson job arrival process** — In queueing theory literature, arrivals are commonly modeled as Poisson processes as this conforms closest to reality. We do the same here as we do not consider cases where there are sudden discontinuous changes to the arrival rate.
2. **Fixed trusted computation overheads** — Literature shows that the relationship between job/data volume and processing overhead is neither linear nor easily characterised [49] we use a realistic fixed overhead however to simplify these simulations. We capture the true relationship in our practical implementations in § 3.2.2
3. **Uniform producer-queue propagation delay** — Depending on factors such as geographic location and network conditions, different hosts might have different latencies to different queues. In practice, the differences between these on a per-consumer basis are negligible **[Citation Needed]** and we therefore ignore these and treat them as part of the service rate intrinsic to the consumer.

4. **Blocking consumer job execution** — In practice, a consumer may be capable of multi-threading and/or working on more than one job at a time asynchronously. We consider the worst case, which is if each consumer can only execute a single job at once.

5.

6.

Our sensitivity analysis shows that the parameter that has by far the most influence on the outcomes of our simulations is the job distribution strategy. Other parameters had little effect. Tables 3.2 and 3.3 illustrate this.

Parameter	S1	S1_conf	ST	ST_conf
hostCount	0.001900	0.004855	0.000315	0.000320
jobCount	0.002016	0.005753	0.000401	0.000292
privateCount	0.009531	0.014117	0.003168	0.002777
strategy	0.851591	0.223900	1.052321	0.188936
arrivalProcess	0.001604	0.003834	0.000224	0.000232
meanArrival	0.004068	0.014289	0.002588	0.001077
queueLenRenewal	0.001711	0.003393	0.000175	0.000198
selector	0.064503	0.101361	0.145021	0.098322
estimator	-0.000355	0.004304	0.000221	0.000187
estimationWindow	-0.001867	0.004473	0.000244	0.000229

Table 3.2: Results of Sobol sensitivity analysis; first-order sensitivities, total order sensitivities, and confidence intervals

hostCount	jobCount	privateCount	strategy	arrivalProcess	meanArrival	queueLenRenewal	selector	estimator	estimationWindow
0.0000	-0.2069	-0.1824	-0.1780	-0.2143	-0.2204	-0.2091	-0.2113	-0.2081	-0.2046
0.0000	0.0000	-0.3395	-0.1873	-0.3819	-0.3825	-0.3809	-0.3226	-0.3799	-0.3790
0.0000	0.0000	0.0000	-0.5026	-1.6796	-1.6244	-1.6820	-1.3792	-1.6823	-1.6960
0.0000	0.0000	0.0000	0.0000	5.2240	4.7283	5.5218	12.7951	5.5375	5.4983
0.0000	0.0000	0.0000	0.0000	0.0000	-0.3024	-0.3182	-0.2568	-0.3128	-0.3154
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-1.1406	-1.2644	-1.1279	-1.1031
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0822	-0.0612	-0.0714
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-8.3210	-8.4123
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0131
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 3.3: Results of Sobol sensitivity analysis; second-order sensitivities

Comparison of strategies

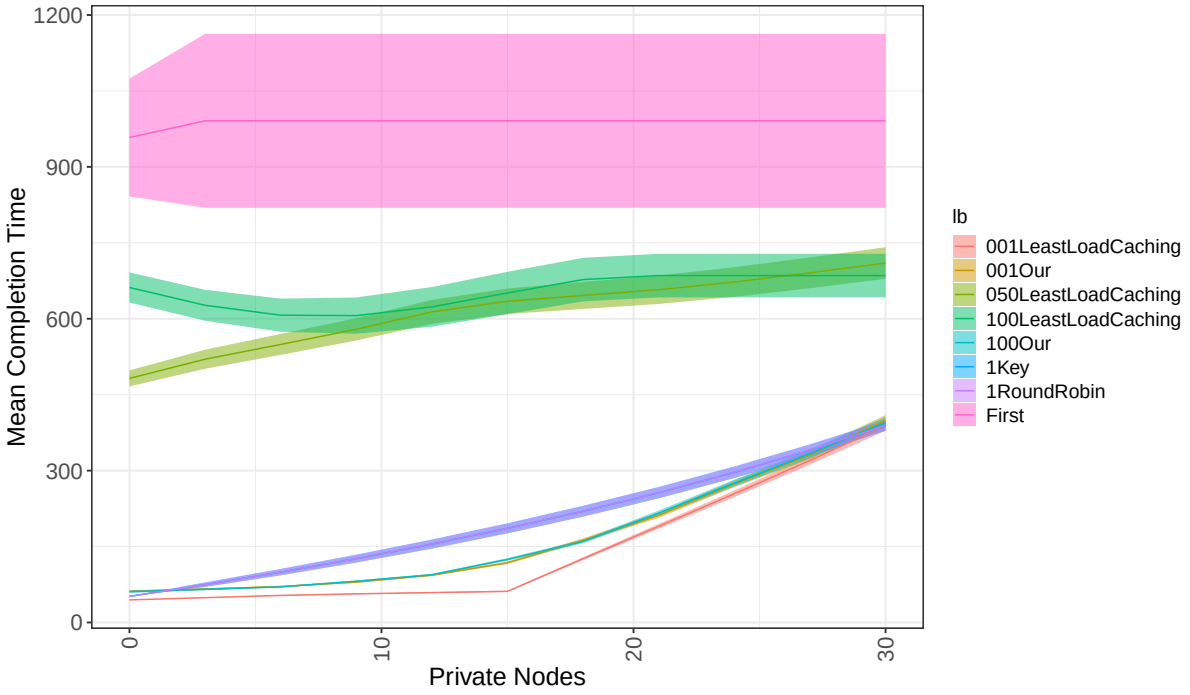


Figure 3.13: A comparison of different job distribution strategy timings as ratio of nodes shifts to private

Figure 3.13 shows the distribution of mean completion times (0.95 confidence interval) when changing the ratio of private consumers from 0 to 30. The pattern is largely the same for mean arrival rates that are greater than mean service times. We also consider that load balancers may be operating on stale queue information and our method takes mitigates this.

Here we show that there's a point at which adding additional private consumers has a diminishing negative effect on performance, and this effect is entirely mitigated by our custom load balancing strategy.

Figure 3.14 shows how queue backlog changes over time during the simulation. This plot serves to simply offer a temporal perspective to our results.

3.3.3 Summary and discussion

We have now demonstrated how we can build in a notion of privacy awareness to augment our job distribution system and others like it. We have done this by first establishing an empirical definition of privacy, however our system is compatible with arbitrary definitions. We have evaluated the privacy vs utility trade-off for our notion of privacy, and

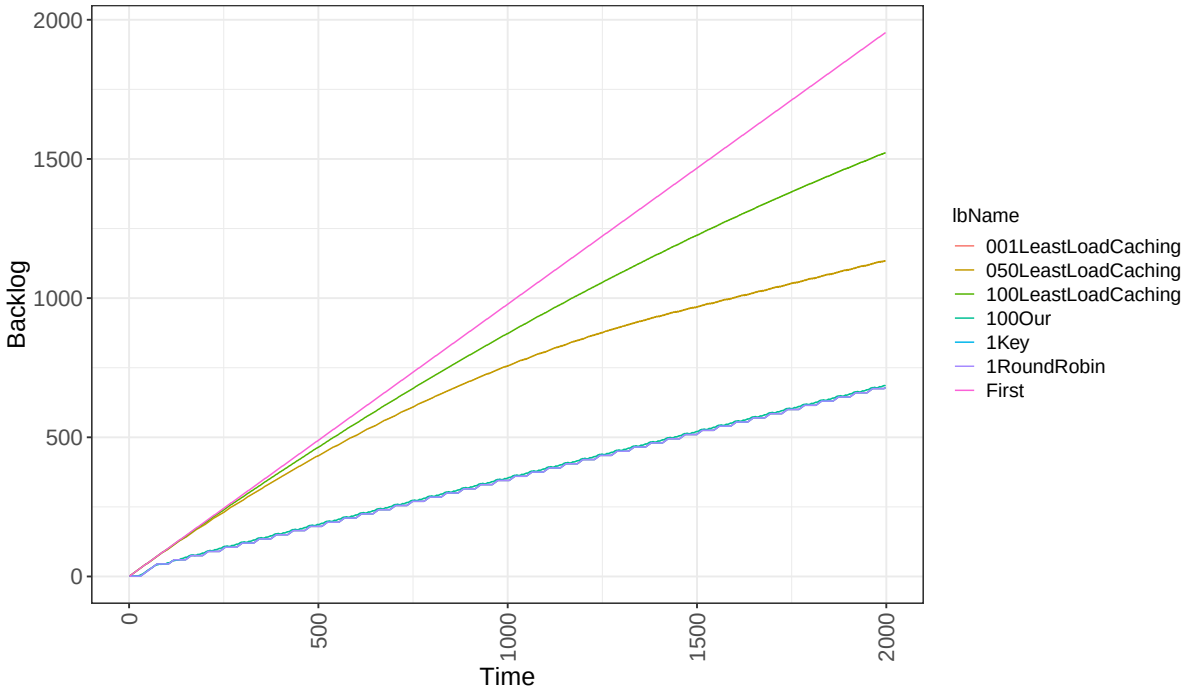


Figure 3.14: A comparison of different job distribution strategy backlog drains as ratio of nodes shifts to private

then shown how applying a privacy to job distribution can be done in a way that does not impact performance, as we make up for it by exploiting the heterogeneity characteristic to improve performance, cancelling out any negative effects.

Chapter 4

Edge computing in the home

4.1 Overview

In §1 we discussed in length where the data sources that we process actually are. A large percentage of the personal data that we emit today comes from IoT sensors and client devices. Many of these, especially IoT devices, are in the home.

It is therefore important to consider the case of moving computation as close to these sources as possible for the sake of privacy and performance. In this chapter we do just that — we start with a detailed examination of IoT sensor data and use cases, and the home network. The paper from which our home network analysis stems has surprisingly generated the most discourse as other have wanted to attempt similar experiments. Then we present our solutions for adding privacy-awareness to these systems (specifically access control and discovery systems) and implement and evaluate these over a real platform.

4.1.1 IoT analytics

Early on in our research, we organised a week-long hackathon[1] where we challenged PhD students to compete in the development of end-to-end home IoT applications integrating sensors connected to Arduino boards (simulating a home environment) and smartphone sensors made available using SensingKit [64].

In order to enable this challenge, we built Representational State Transfer (REST) servers that accept streaming sensor data, allows the different groups to query and retrieve these in various ways, as well as visualise the data in the browser as real-time plots. We

[1] `http://cis.eecs.qmul.ac.uk/IoT2016.html`

also build example smartphone apps to aid in development. Our code is open source and available on GitHub under `yousefamar/cis-rest-server` which links to other relevant repositories.

This hackathon allowed us to make some key observations with all the collected timestamped data that was stored by the servers over the course of the event. Insights on the volume, frequency, and format of the data, coupled with the application-side granularity requirements of said data, showed us that we did not actually need to store data at that level of fidelity for the applications to still preserve their utility. It would make applications more privacy-preserving, and indeed more performant, if we introduced a component into the pipeline that filters the data at the source.

We have already developed a system that can do this, which we delineated in §3.3.1. In this chapter, we apply this system to access control mechanisms to control and attenuate the flow of data, and provide the same privacy benefits that we demonstrated before, at negligible performance overheads.

4.1.2 Comparison of access control and discovery systems

While we have already established the advantage of using existing standards, as opposed to rolling our own, there are several reasons why we chose these particular standards rather than others. This section details the primary reasons for implementing this system by building on a combination of Hypercat and Macaroons.

Hypercat catalogs descrbe assertations on resources in a subject-predicate-object style similar to Resource Description Framework (RDF) triples. It is however preferable to other methods of cataloging and describing resources, such as RDF documents, for a multitude of reasons:

- Use of the JSON format, instead of e.g. XML, is better suited to a web ecosystem. Beyond HTML for marking up web pages, it is standard for REST APIs to pass data formatted as JSON. As the standard scripting language of the frontend web (and increasingly the backend also, in the form of V8-powered NodeJS) JavaScript has built-in functions for parsing JSON. As JSON maps perfectly to JavaScript objects, it is much quicker and clear-cut to handle JSON data.
- Hypercat as a standard has very recently joined the British Standards Institution, indicating that it is here to stay. Using a well established standard not only avoids reinventing the wheel, but contributes to future-proofing the system.

- Hypercat in itself does not define naming schemes that relations in catalog items need to use. Instead, Hypercat allows the definition of pointers to naming schemes. These can be set by third parties and are subject to be changed by the "invisible hand" of the market, depending on what what ontologies developers converge on.

- As already mentioned, real-world services especially in this space (IoT and personal data) are more inclined to use RESTful APIs for simplicity, instead of Simple Object Access Protocol (SOAP) APIs. This influenced the development of Hypercat, and is one of the benefits that Hypercat contends over formats like WSDL, when it comes to interoperability in an ever-evolving web.

In contrast to Hypercat as a standard, there are not as many existing standards for authorisation tokens with cryptographic properties as there are for describing resources. Our system using Macaroons has a number of advantages over using other mechanisms of delegated authorization such as OAuth 2.0 and OpenID. It is important to note that our use case differs in additions, so using different mechanisms would necessitate rolling our own implementations of functionality already covered by macaroons, on top of them. In fact, macaroons can be used in conjunction with OAuth, but doing so does not serve our purpose. The Google Research paper that first specified macaroons [67] goes into detail on why macaroons as credential systems are superior to others such as cookies and SPKI/SDSI, and this is outside the scope of this paper. The advantages for using macaroons in our system include:

- Macaroons are like signed cookies, in that after sharing a unique key once, you no longer need to query the minting party for any information, as that key (used to sign the macaroon) can be used to validate it. This is necessary for scaling the system, where the token-minting component (the arbiter) is separate form the many stores that verify these tokens.

- Similarly, permissions are encoded within the tokens. Stores can therefore know the extent of what a bearer is able to do simply by inspecting their token.

- Macaroon caveats are stackable, and map well to our route-based permissions system described earlier.

- Macaroons allow delegating authorization in a secure manner by design. No other standard covers this. This lends itself well to the Databox ecosystem, where components can be distributed across many devices, each with their own set of stores and access to data sources and sinks. More permissive macaroons can be narrowed down repeatedly in a hierarchical fashion as they are passed on to other hosts.

- Since our system can span across distributed hosts, the inherent cryptographic properties of macaroons that prevent tampering are well-suited for use across networks.

Besides being individually suited to our requirements, these standards also mesh very well with each other. The combination of the two is simple because our route-based permissions system is well suited to modern web API development (REST, JSON, etc) and straightforward to encode into macaroons. Meanwhile, Hypercat similarly targets the same web technologies, and the specifications keep any authorization mechanisms that may be applied to Hypercat completely open. Hypercat only specifies the way by which tokens are passed, namely through Basic Auth, or a particular HTTP header, but no more. This means that our serialised macaroons are trivial to integrate into Hypercat, and the accessibility of catalogs, catalog items, and the visibility of catalog item metadata, can be all be – and are – controlled by the same system.

4.2 An analysis of home networks

There is an increasing presence of internet-connected devices in our homes, and yet we see an alarming rise in data thefts, security threats, and privacy breaches through these unregulated, uncertified, and often insecure devices.

With the recent discovery of the KRACK replay attack on devices using WPA2 [118], the risks of unsecured communication within a network have once again surfaced to the public eye.

Similarly, with standards slow to catch up, manufacturers of networked devices have been making assumptions about the protocols they use, relying on undocumented behaviour. Standards to combat these issues do exist. For example, HTTP/2 requires encryption by default, blacklists insecure cipher suites, and is still faster than HTTP/1.1. Yet manufacturers are slow to adopt it and rely on legacy hardware and software.

TLS 1.3 uses ephemeral keys, making it more difficult for an attacker to decrypt traffic. This has seen pushback [102] as some heavily regulated enterprises (e.g. banking) rely on using static keys to decrypt and monitor internal traffic to meet both security and visibility requirements. This emergent use case could not have been predicted and solutions that will satisfy contexts with different concerns — such as home vs enterprise — are therefore slow to be established.

Along the same lines, some imminent standards, such as DNS Over HTTPS (DOH), seek to solve many privacy issues and prevent service providers from discriminating be-

tween different kinds of traffic. This would mitigate DNS-based internet filtering by ISPs/governments, as well as greatly enhance user privacy.

There are many potential measures that can be taken to mitigate the issues we discuss in this paper, however they have only very recently begun to be explored [98].

From an IoT perspective, in order to understand how to contain these devices, we need to first understand their behaviour in-the-wild and in realistic deployment scenarios. In order to secure the home of the future, it is critical to be able to automate the process of fingerprinting their network behaviour such that threats across arbitrary devices can be identified and mitigated.

We aim to understand the overall IO behaviour of devices in an average IoT-enabled house. Our intention is to understand the protocols used, data volumes, types, data rate and transmission frequency, etc. Our data should enable us to assess the potential costs of these devices in terms of bandwidth and their privacy and security threats.

Doing so will allow us to develop systems that maximise privacy and minimise leakage by default, and providing an interface for users to be able to understand, monitor, and control the flow of personal data in their homes [46].

We demonstrate simple yet significant measures to circumvent attempts at securing devices and protecting privacy, such as using passively observed API keys to hijack control of smart light bulbs, or tracking Apple devices despite MAC address randomization.

4.2.1 Set-up and dataset

Devices

Figure 4.1 provides an overview of the IoT devices in our home testbed. The devices we used are as follows.

① Foobot Air Quality Monitor

② Netatmo weather station and environmental sensors

③ Amazon Echo

④ Apple iPhone 5 running iOS 11

⑤ Apple iPad pro 10.5” with cellular and WiFi connectivity

⑥ Apple Macbook Pro 13” with touch bar

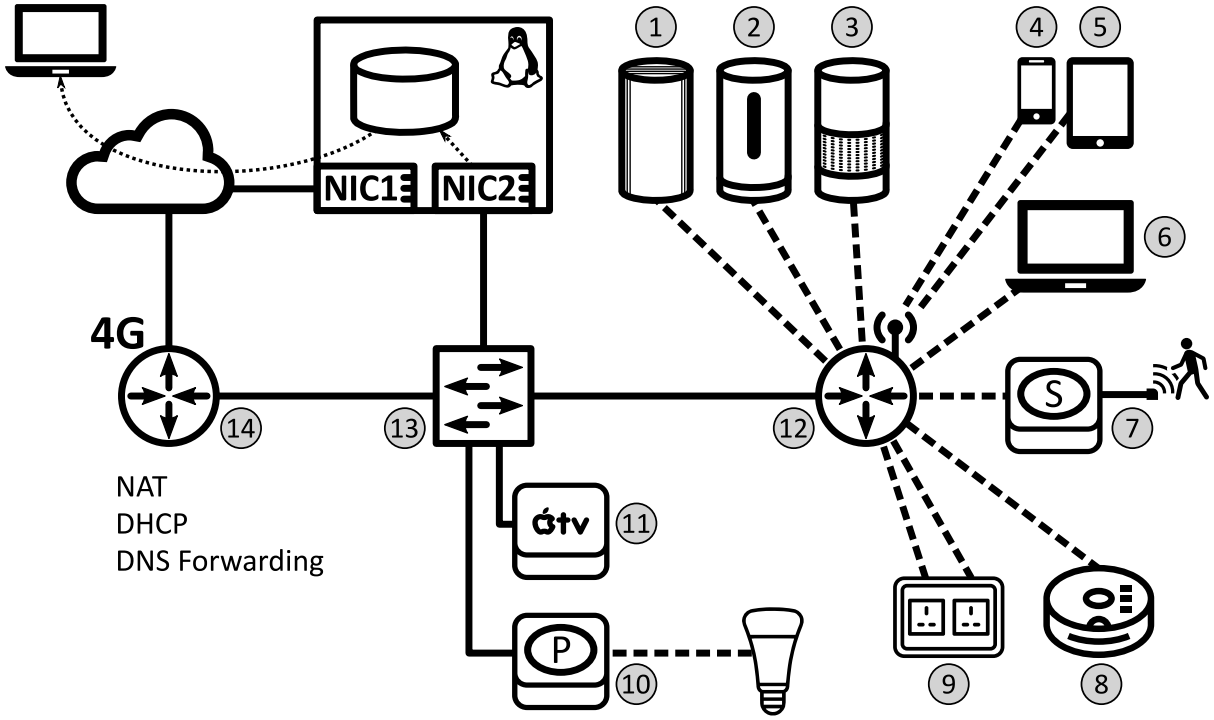


Figure 4.1: An Overview of the Home IoT Testbed

⑦ Samsung SmartThings Hub with a presence sensor

⑧ Neato Botvac vacuum cleaner

⑨ 2x TP-Link smart plugs

⑩ Philips Hue Bridge

⑪ Apple TV

⑫ Ubiquiti Access Point

⑬ DLink Switch

⑭ TP Link MR6400

These include a variety of general-purpose home hubs (③, ⑦), as well as device-specific hubs (⑩, ⑪) that are one step away from the devices they actually control, such as several Hue LED bulbs.

We also have a number of consumer electronics connected (④, ⑤, ⑥), while our two smart plugs (⑨) can accommodate any additional offline devices. Finally, we have IoT devices that connect directly to the router (①, ⑧, ②).

Set-up

This section describes how we set up our home IoT testbed. Our measurements and analysis is conducted on network traffic while all devices are idle. We connect an L2 switch to a 4G router running NAT, DHCP, and DNS forwarding to Google's DNS. An access point is connected to this switch to which all but two of the IoT devices are connected. These two have a wired connections directly to the from two of our IoT devices.

We capture all traffic by mirroring all ports to one connected to a Linux box with two NICs. Traffic is captured via one NIC with TCP/UDP disabled through tcpdump and stored on disk. Traces are then retrieved separately through SSH via a second NIC connected to the internet. All data must pass through the switch (both internal and external) and thus all packets are mirrored on the switch port to the Linux box and therefore all packets are captured.

Data and analysis

We continuously captured packets for 22 days before performing our first analysis of this data. We wrote a set of scripts to perform our analysis, and are making these scripts publicly available [4]. To analyze network behavior on a per-device basis, we split the combined trace by MAC address. We also used DNS and DHCP logs to help find hostnames that correspond to MAC addresses by looking at mappings of IP to MAC address and IP to hostname over time. This is useful especially for devices that randomize their MAC addresses, such as the iPad.

For all other statistics, we fed the traces through the Bro Network Security Monitor [104] and ran custom as well as existing scripts on Bro logs.

4.2.2 Observations

Device setup and interaction

The devices were set up using the manufacturers apps and set up with Apple HomeKit where possible. If a device could work with Alexa then it was configured to do so (Foobot, Neato Botvac, Philips Hue, and TP-Link plugs).

When setting up devices, it is not uncommon to go through a Bluetooth-like pairing flow (e.g. Apple TV). Some devices require WPS-like physical interaction through a button press (e.g. Philips Hue Bridge), however we show that this can be circumvented.

Table 4.1: Comparison of device interactions.
physical = ⧫, *encrypted API* = •, *unencrypted API* = ∘,
encrypted authorization = ■, *unencrypted authorization* = □
check

Device	Configuration	Interaction
Foobot Air Quality Monitor		■/□
Netatmo weather station		■
Amazon Echo	⧫	■
Samsung SmartThings Hub		
Apple TV	⧫	•
Philips Hue Bridge	⧫	□ *
TP-Link smart plugs	□	•
Neato Botvac	□	∘

Previous work has shown that the secure setup, pairing, and configuration of devices can be done securely in way that does not impact user experience negatively, so arguments for compromising security in favor of usability are weak [13].

Some devices, such as the Hue Bridge or the Neato Botvac communicate via plain HTTP with other devices and the outside world. This interaction can be monitored, device behaviour inferred, and in the case of the Hue Bridge, API keys can be extracted and the device hijacked.

Table 4.1 summarizes how each device is configured on setup and the means of interaction during operation. These have implications for security and in some cases, such as with the Philips Hue Bridge, there are vulnerabilities (discussed in the next section) that negate certain security measures.

Learning from traffic

Identifying devices

While for most devices, the first three MAC address bytes are enough to identify a vendor, and thus make a reasonable assumption as to what the device could be, there are many other methods that we explored in part during this experiment that can be used in conjunction.

For example, the Foobot's MAC address points to "Shanghai High-Flying Electronics Technology Co., Ltd" a WiFi module manufacturer. This tells us very little about what the device could be, but if we look at the DNS requests it makes, most of which are A record queries for api.foobot.io, we can build a behavioural profile for this device, and use it to identify others like it.

Similarly, consumer Apple devices can randomise MAC addresses to curb tracking,

yet there are ways to track these devices regardless or even reveal their true MAC address [119]. We were able to track an iPad connecting to our network by combining DHCP and DNS logs to ascribe the iPad's local hostname to all eight of the MAC addresses it used.

Monitoring devices

While encrypting web traffic protects a user's data to some extent, there are still other risks that are not always obvious. For example, using a proxy will not always curb DNS leaks unless configured for remote DNS. A lot of information can be gleaned from DNS traffic, though fortunately efforts to protect this information (mainly from ISPs) are increasing. Recent commits to Android indicate that Google plans adding DNS over TLS to the ubiquitous operating system [23].

The main issue however is that some devices still use plain HTTP for some or all of their communication (figure 4.2). Any device on the network can passively observe metadata and control requests coming from these devices next to the usual unencrypted browsing done by mobile devices. This can be a significant privacy risk, even for seemingly innocuous data, such as light bulb states. Inferences such as presence in rooms or homes can be made just by inspecting the responses to state requests, e.g. in this case, the periodic requests by the Amazon Echo to the Hue Bridge. Other information such as who was the last to control the lights and when can also be extracted from these requests.

Controlling devices

Some devices have rudimentary security, such as the Hue Bridge, which requires a physical button press to register a new device. However, when keys and credentials are transmitted over plain HTTP, as they are in the case of the Hue Bridge, any device in the home network can sniff these, making the bridge susceptible to replay attacks and the keys can be hijacked to make other API calls.

We tested this by extracting the iPhones API key from our network trace, and making API calls to query state and control lights from a separate laptop with a different MAC address. The requests remained valid even after several weeks during which the bridge was unplugged. This indicates that the key is persistent, has no short timeout, and not bound to any device by IP or MAC address.

While a malicious or compromised device, or an intruder in the network, might necessarily have any reason to control someone's lights, devices that emit more sensitive data, or control more critical systems, can be affected by similar vulnerabilities,

Attack surface is of course not limited to WiFi. It has been shown in the past that

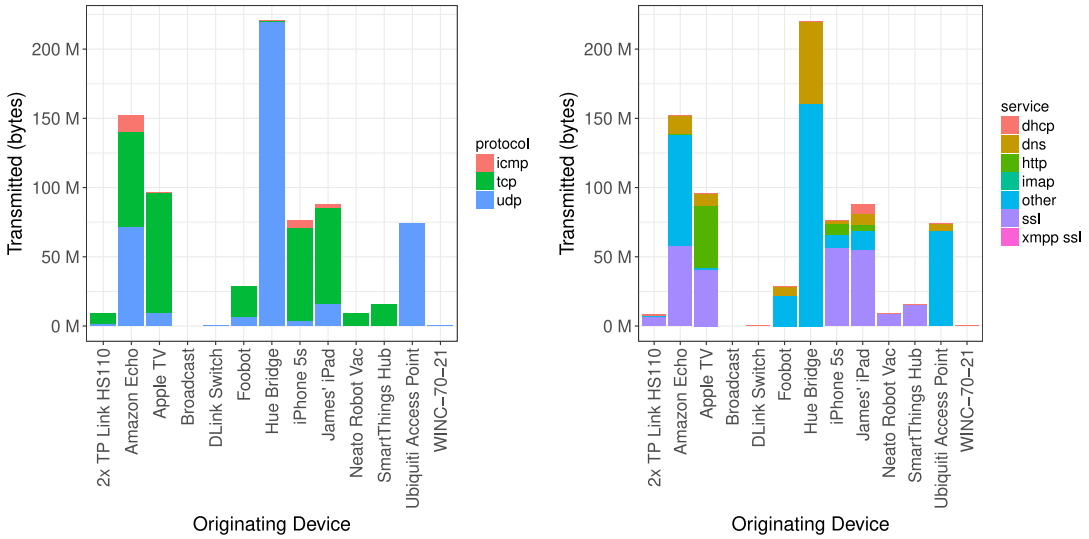


Figure 4.2: Bytes transmitted per device split by protocol and service

a Samsung Smart Things can be compromised through Zigbee [35] for example. As we only captured network traffic, this is outside the scope of this paper.

Global statistics

An initial analysis of our network trace showed that the largest source of data by IP in bytes is an Apple server (25.8%), and correspondingly the top largest sink is the Apple TV (41.9%). These two IPs account for 27.1% and 43.9% of our routers sources and destinations by bytes respectively. This is singlehandedly caused by a large software update, that equally had a significant effect on skewing global port statistics towards HTTP and HTTPS. By volume, just under half of traffic was as a result of this alone.

Figure 4.2 shows the bytes transmitted by device (taken from the IP total_length header field) split by protocol (left) and service (right). Figure 4.3 shows the sum of payload bytes for each connection sent by originator and responder split by internal and external communication, the vast majority of which is external.

This information was extracted from Bro connection logs where each connection has an originator and a responder. Figure 4.2 only shows just the number of bytes an originator sent. Our TP Link MR6400 router accounts for virtually all bytes *responded* at 4.02 GB in total, 1.99 of which is HTTP, 1.85 SSL, 0.09 DNS, a major part of which can be attributed to the Apple TV update. The router was left out of figure 4.2 as it dwarfs the remaining devices being responsible for the most originating bytes at 778.73 MB.

This is especially visible in the inset plot in figure 4.4 where the Apple TV update

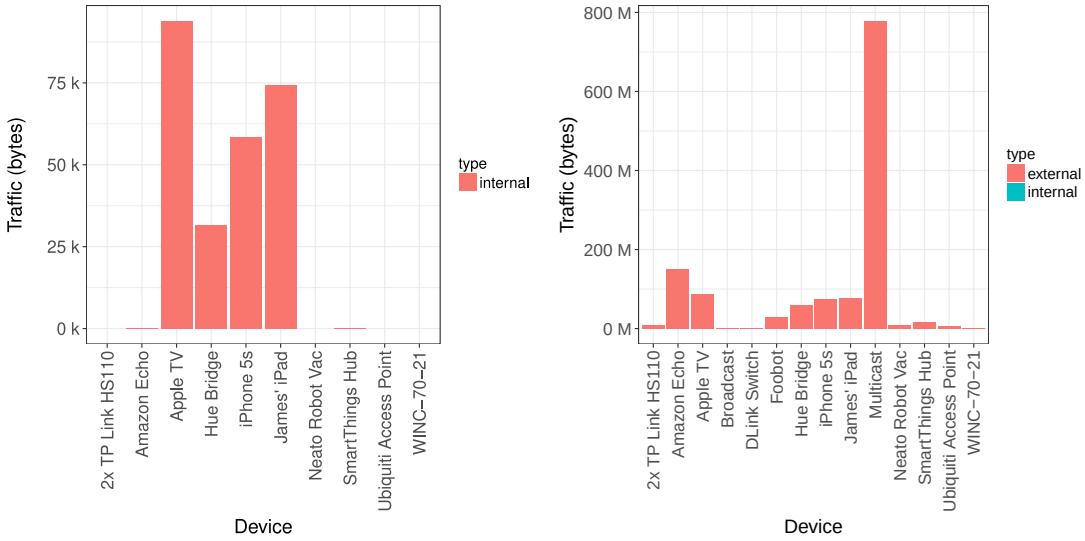


Figure 4.3: Internal vs external traffic; internal traffic separated for visibility

created large spikes at ①. When these are filtered out, we are left with figure 4.4 proper where most spikes are as a result of iPhone ③ and iPad ④ browsing. The initial spikes at the start of the trace ② are caused by traffic during configuration and is discussed in the next section.

Statistics by device

Router

Other than the Apple TV update and some MQTT pings, virtually all traffic originating at the router were from SSDP packets. We can tell this alone by destination, which was the SSDP multicast address. The Search Target headers contain device and service schemas that are typical of routers, for example, InternetGatewayDevice, WANDevice, WANCommonInterfaceConfig, WANConnectionDevice, WANIPConnection, WFADevice, WFAWLANConfig, and Layer3Forwarding. However no devices request UPnP descriptions from the URLs in the location headers.

Hue Bridge

The Hue Bridge's behaviour is particularly interesting, as the vast majority of its traffic is consistent DNS and SSDP, after a brief burst in the beginning during configuration. This is primarily (787552 requests) to www.ecdinterface.philips.com, however it does not attempt to connect to this host.

The Hue Bridge transmits bursts of just over 2 KB of basic device SSDP traffic

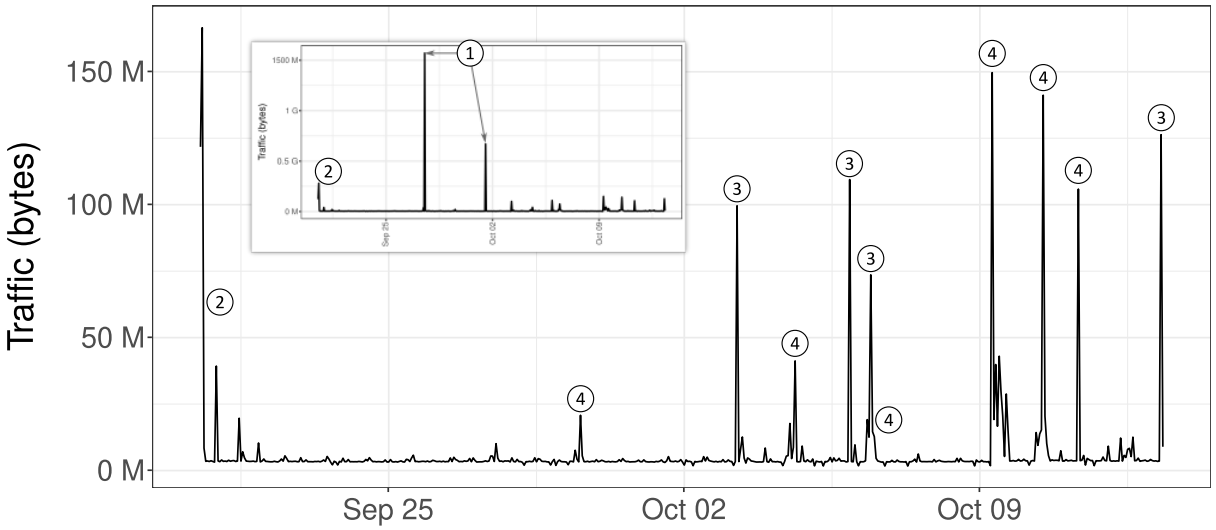


Figure 4.4: Hourly aggregates of frame lengths over time with spikes from Apple TV ①, iPhone ③, iPad ④, and mixed configuration ②

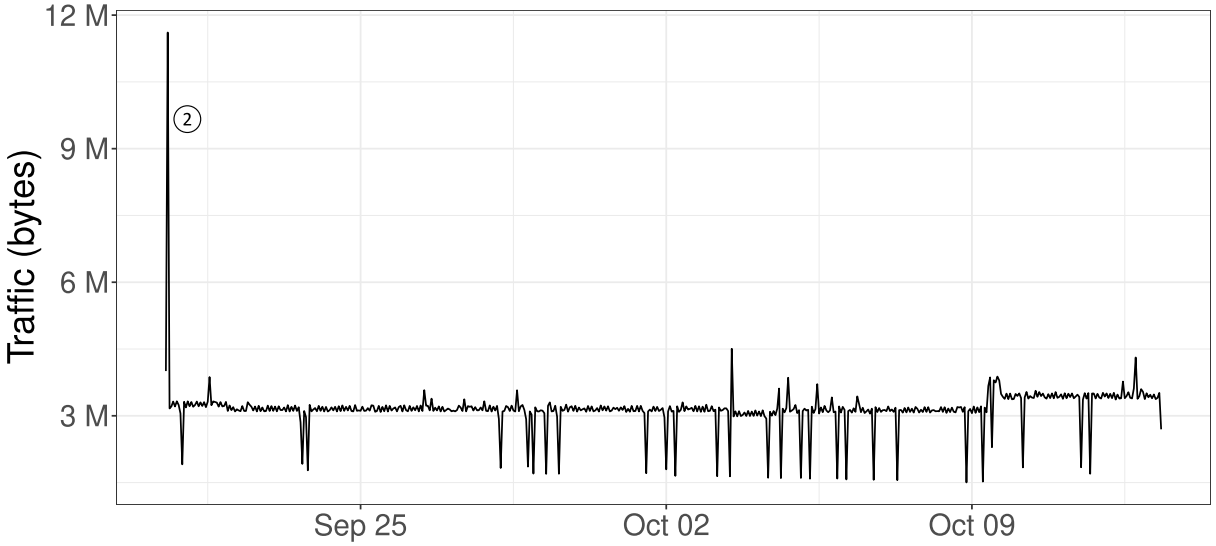


Figure 4.5: Hourly aggregates of frame lengths over time excluding Apple TV, iPhone, and iPad

every 52 seconds, however, unlike the router, its UPnP description is queried by the Amazon Echo. The Echo gets standard metadata from this description, and at thrice the frequency, the Echo calls Hue API endpoints to get the lights' state, including on/off state, colour information, and other metadata. The Echo is the only device that makes API calls to the Hue bulbs, aside from the iPhone during configuration.

Apple TV

Apple TV network activity can for the most part be attributed to the aforementioned automatic software update. Automatic updates can be disabled but are enabled by default. The unencrypted portion of this traffic are primarily spikes while requesting image and video thumbnails from Akamai Technology CDN servers.

These include film cover art and app icons, so viewing habits and installed apps can be inferred indirectly by examining the frequency certain thumbnails are downloaded. There exists therefore the potential for embarrassment on, for instance, cultural or religious grounds.

When looking at purely the number of connections by port, DNS has the highest at 4480 followed by SSL at 4434, and multicast DNS at 3580. From the DNS requests, we can see that the most frequent is a local HomeKit hostname, followed by Apple's Bonjour Sleep Proxy (common across any Apple device), and a number of Apple application and time servers (time-ios.g.aaplimg.com, time-ios.apple.com, itunes.apple.com, init.itunes.apple.com, xp.apple.com, play.itunes.apple.com, etc). There are no other discernible patterns.

Amazon Echo

Aside from the traffic corresponding to Hue Bridge API and UPnP requests previously discussed, all HTTP traffic is encrypted. The Echo communicates with several Amazon servers. The DNS logs hint on the Echo's behaviour, with 4392 requests for device-metrics-us.amazon.com, 1210 for dcape-na.amazon.com, 538 for pindorama-eu.amazon.com, and 46 for softwareupdates.amazon.com, among several others. Interestingly, there were also 490 requests for www.meethue.com; the model URL listed in the Hue Bridge's UPnP description.

The vast majority of DNS requests were very unusual – 69192 for www.example.com, 69120 for www.example.net, and 69086 for www.example.org. Other users have observed this, however we can only speculate what the point of these requests are, since the Echo's interaction with these hosts is limited to the TCP three-way handshake.

By number of connections, 29796 NTP connections follow DNS, with the most at 44973, followed by HTTP at 22090 connections. There were also 6778 ICMP connections made from port 8 on the Echo to port 0 on the router.

Neato Botvac

In figure 4.2, this device's total transmitted bytes are relatively small since it was turned off between 2017-09-22 01:20 and 2017-10-04 15:50. The vast majority of traffic is TCP; a baseline of SSL traffic with mostly TCP Keep-Alive packets. It communicates mainly with two AWS servers in the cloud. Browsing to these shows that these are Neato's RESTful API servers; "Nucleo" servers with version 1.11.1 of a "Neato Robotics Message Bus.". Other than that, there is minor NTP traffic.

iPhone

Besides using device apps for configuration and setup at the start, subsequent iPhone traffic is characteristic of normal browsing behavior. There are random spikes of HTTP/SSL traffic to email, social media, and news websites, as well as the common Apple servers (akamaitechnologies.com CDN, etc). Some requests are to Amazon, attributable to Alexa.

Macbook Pro

This was the first device that had Dropbox traffic. It was only really active during the first two hours of capture for configuration, and had barely any traffic at all after that, except short periods of primarily SSL, some HTTP traffic. Coupled with DNS requests to e.g. Facebook and Google servers, this is indicative of light browsing.

Some of the HTTP traffic was requesting `generate_204` characteristic of network portal detection for logging in to WiFi, otherwise JSON headline dumps from news websites and some images.

Here too we have the usual Apple traffic, including sleep proxy and HomeKit, as well as Hue Bulbs interaction. The Macbook also had some encrypted communication to a second Netatmo server (b90.netatmo.net as opposed to b91.netatmo.net).

4.2.3 Summary and discussion

In this work, we analyse network traces from a testbed of common IoT devices, and describe general methods for fingerprinting their behaviour. We then use the information and insights derived from this data to assess where these privacy and security risks

manifest themselves, as well as how device behaviour affects bandwidth and power consumption.

We published scripts to simplify doing this form of network trace analysis and identified several areas of contention when it comes to privacy such as Philips Hue Bridge security holes and simple circumvention of MAC address randomization. The main takeaway from this work is to begin to provide an awareness on the behavior of common home IoT devices to mitigate privacy risks.

Our initial analysis was on a simple case where all IoT devices are in their idle state. We plan to repeat this analysis for several additional scenarios and under different conditions.

Through observing behavior when users directly interact with these devices at home and remotely (e.g. through an app) we seek to ultimately draw inferences from the traffic, such as discerning human activity and evaluating the privacy risk of being able to do so. Similarly, we plan to lay the groundwork for identifying important considerations when building integrated IoT/home hub systems from a network security perspective.

4.3 Case study: the Databox platform

As a precursor to our research, we designed and built a functional prototype of the Databox platform. For brevity, we only describe the implementation details of the parts of this platform that are pertinent to the research context. Other work, such as on example applications and frameworks for streaming mobile sensor data, are left out intentionally.

4.3.1 Overview

TODO: Update outdated information

A Databox is a home IoT hub, that is supported by cloud services. It makes diverse personal data sources (from online/social to IoT to mobile) accessible and provides runtime APIs for interfacing with that data.

A Databox is a personal networked device in the form factor of a router or home hub, located in a user's home as a nexus to their digital life. It is supported by an ecosystem of open-source services, both local and in the cloud, and enables individuals to manage their data, and to provide other parties with controlled access to their data.

Data sources can broadly be split up into three potentially overlapping categories: on-

line data (email, banking, social media, etc), mobile sensors (location, motion, pedometer, etc), and IoT devices (light, temperature, energy, smart devices like TVs, fridges, etc).

Individuals about whom data is collected from these sources (henceforth referred to as data subjects) can, through a dashboard interface, have a complete overview and fine-grained control of where their data is coming from, and where it is going to.

Individuals and organisations wishing to process data (data processors) can then ask subjects for permission to do so. The Databox provides APIs that the processors can take advantage of to run their routines in a sandboxed environment on the device itself. If necessary, and user willing, the Databox can emit any results of the executed analytics back to the processors.

These APIs that support a range of uses, ranging from privacy-preserving detailed analytics (e.g., on mental/physical health) to aggregate population surveying and statistics.

It is important to note that – unlike some of the related work previously mentioned – the Databox is *not* a silo but rather a platform through which data can be accessed and processed locally. Some of the issues pertaining to data access are however shared.

For both technical and privacy reasons, data processors are incentivized to access and process all data within the sandbox environment that is a Databox, and only emit results to the outside world. By bringing the analytics to the data, rather than the data to the analytics (in the cloud), Databox reduces the risk of inferences – let alone privacy breaches – to an exceptional degree.

Databox components run in isolated containers with limited, encrypted communication. Next to components that perform administerial functions, there are three core components.

Drivers, which can be developed by third parties, interface with outside data sources and query data. These write data into *stores* which provide a common API and access control for other components to access this data. Stores are system components and launched alongside drivers. Finally, *apps* (most of which will be written by third-party developers) can be launched with permissions to access certain stores, and thus certain data. Once these have finished their processing, they may be allowed to emit limited data to the outside world through an export service. Apps are packaged alongside a manifest file which, among other metadata, lists the required and optional permissions that an app may need.

Architecture

An early Databox software prototype – as well as supplementary software and tooling – at the time of writing this document, is already up and running at a near *Minimum Viable Product* stage. All source code is currently available at github.com/yousefamar but will likely be transferred to live under the DataboxProject organisation.

The repositories *databox-minimal-dashboard* and *databox-data-arbiter* both contain "firmware" code, while *databox-app-server* is for the remote app server, as the name suggests. For mobile sensing using SensingKit [64], the repository *cis-rest-server* contains a precursor for streaming and visualising sensor data from mobile phones.

Currently, two repositories serve as high-level pseudo unit tests and example apps. The *databox-hello-world* repository contains a minimal app to demonstrate manifest file format/use, Dockerfile options (explained shortly), and utilization of serving a simple UI to a dashboard. The *databox-twitter-wordcloud* repository demonstrates use of a native data source and data processing at different levels – sentiment analysis is done at the app container level, and word cloud generation at the client level. It also serves to show how client-side data leaking can be, and is, programmatically blocked as is discussed in this section.

A useful picture to have in mind when considering individual components is the following figure 4.6. The most important aspects of this diagram are described below.

After weighing the advantages and disadvantages of using Docker [29] vs Unikernels [117] for app deployment, Docker was collectively decided upon out of maturity and versatility. Beyond isolation, Docker also simplifies distribution and deployment. This is discussed in more detail below.

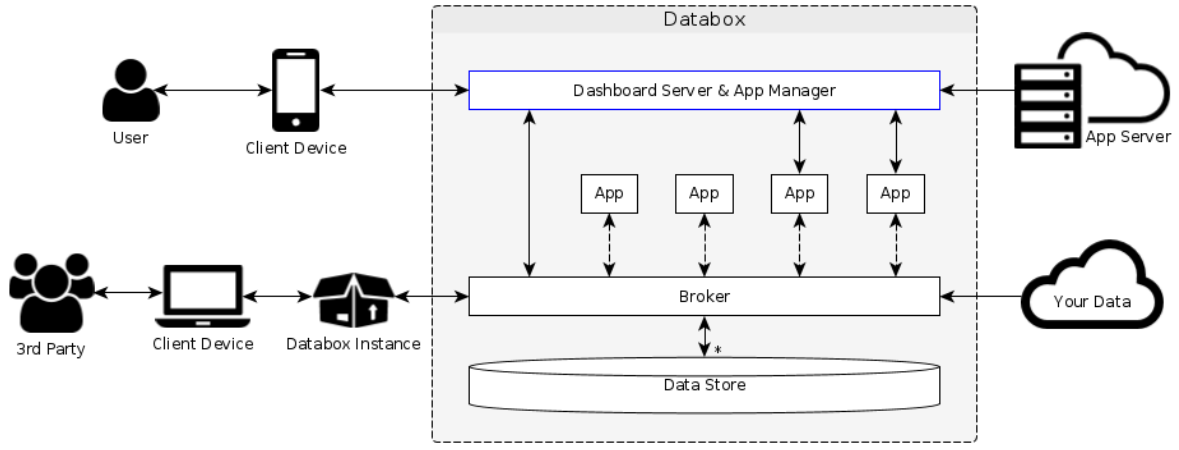


Figure 4.6: A High-level Overview of Databox Components

Dashboard and container manager

The container manager which doubles as a dashboard server is the only Databox component that does not run as a Docker container. On first startup, the dashboard program is launched alongside a Docker daemon, and it subsequently pulls the needed images from a remote Docker registry and launches them. These containers are not dependent on a dashboard instance running but are merely monitored and controlled by it.

The dashboard makes use of the Docker Remote API to control arbiter and app containers. It pulls newer Docker images from a remote registry, and maintains a WebSocket connection to a *Client Device* in order pass notifications of any Docker events to the front-end and thus indirectly the user. Here the client is simply a web browser or a browser wrapper (e.g. Node-Webkit [99] or Electron [33] for standalone desktop clients, or WebView wrappers for mobile devices).

When the user launches an app via the client, the server (dashboard) used the Docker Remote API to launch the corresponding app container with proper configuration. This includes port bindings for apps that serve their own UIs, and Docker container links (dotted lines in figure 4.6) to allow app containers to communicate with the arbiter container. An app container is also named after the app name; this means that two instances of the same app cannot be launched on the same Databox since the name collision will not allow it, nor would they need to be.

The dashboard provides several views. The app "store" is rendered client-side as part of the dashboard, with the store server simply providing REST endpoints. Once an app Docker image is pulled from the registry, it can be launched from a list of loaded images. Of course the dashboard also gives you a general overview of Databox functions. The *arbiter* UI is viewed through the dashboard and contains a list of available data sources and allows you to configure them (e.g. authorise the Twitter app).

Similarly, the dashboard allows you to view app GUIs. These are also plain HTML/CSS/JS served by app containers and proxied through the dashboard interface. To do this, the dashboard, on launching the arbiter container or an app container, will automatically map a free port to the internal Docker port 8080 if the app manifest specifies that an app serves a UI, starting at port 8000. The exception is the arbiter container which serves to 7999 and is always mapped to 7999 externally (see figure 4.7). The dashboard in turn proxies requests to `/app-name-here` to the correct port on localhost. The dashboard embeds app UIs into sandboxed iframes [1] but alternatively these could be displayed in new windows.

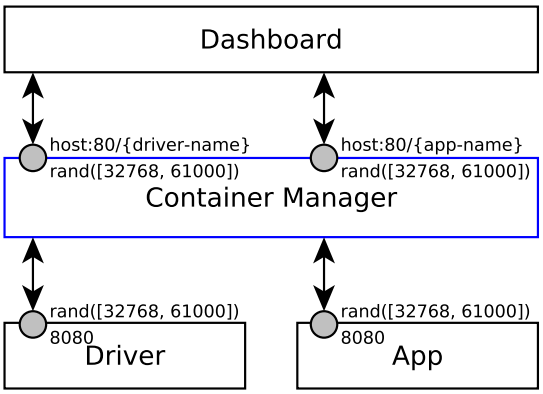


Figure 4.7: Automatically mapping exposed Docker ports to free external ports

Apps, once authorised through token-based authentication, can make use of the Docker container link to the arbiter container to access your personal data. Apps are then free to process this data within the isolation of their container, but can also do some processing client-side on the user's device. This is mainly for processing in the context of data visualisation that needs to be done on a client device for the sake of latency (e.g. a D3 [24] graph).

This raises a unique issue: how can client-side data leakage be prevented? Even if client-side scripting were disabled by the dashboard, the app UIs could still leak your data by making any sort of request to an external resource. For example an image tag such as:

```
<img src="http://evil-corp.com/?leak=[your personal data here]">
```

To prevent communication with any external server and thus solve this problem by only allowing client code to communicate with the app container, *Content Security Policy* (CSP) is enforced. CSP is a W3C Candidate Recommendation [22] and supported by most modern browsers. CSP can be enforced by returning the HTTP header field "Content-Security-Policy" with any number of directives to essentially control what resources can be loaded from where with whitelists of sources. To prevent loading anything from anywhere except the app container, the `default-src` directive can list only `'self'`.

An additional useful CSP feature is the `report-uri` directive. Using this directive, if an app were to try to make external requests, a report is POSTed to the specified URL. This is normally used for debugging, but for this purpose, it can be used to further enforce accountability and build trust metrics.

The arbiter container

"Arbiter" can be an ambiguous term. The arbiter container serves a number of OS-level functions. First and foremost, access to or from any data source or sink respectively is mediated by the arbiter container. It controls all authentication making its primary purpose *access control* to configured data sources. Another potential function is that of synchronising data stores.

The other main arbiter function is the *access* side of *access control*. The arbiter encapsulates away any interfacing with data sources and exposes simple, standard API endpoints to them. Not only does this allow normalisation of interfacing with different sources, but also the same sources across differing hardware and versions. For instance accessing GPS data from an iOS device versus an Android device or an older iOS device.

App server

The App Server, which can also be thought of as an app "store" server, is relatively simple. It is made up of a number of components as are depicted in figure 4.8 on page 63.

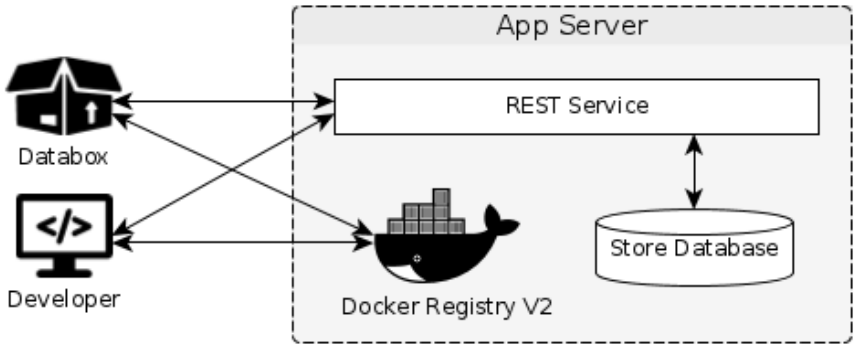


Figure 4.8: A High-level Overview of App Server Components

Most notably, a REST service provides a primary point of control. The API endpoints are listed in the *databox-app-server* repository readme. These are mainly for a simple account system and submitting and listing app manifests and store metadata such as ratings, reviews, downloads, etc. Although the store-front-end is served Databox-side, the app server also serves a simple developer dashboard to facilitate email-verified account creation and submitting apps.

Apps, as Docker images, are stored within a Docker registry. Pushing to the registry is protected through HTTP basic authentication, but eventually authentication will be through an OAuth 2.0-like flow with a separate authorization service returning bearer tokens [30].

4.3.2 Authorisation and discovery over Databox

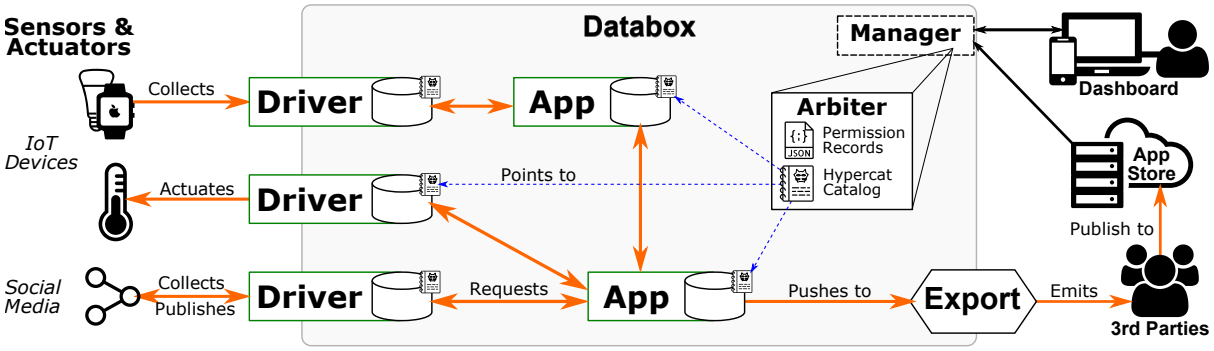


Figure 4.9: Relationships between system and third party (green) components

This section describes our current working implementation of our system. It combines a route-based permissions scheme, with a resource description standard (Hypercat [101]) and signed tokens (Macaroons [67]) for delegated authorization. We call the system component that performs these tasks the *arbiter*.

As a system that needs to be privacy preserving by design, all permission and access are most restricted by default. Only through user interaction should these permissions be able to be made laxer. For components that are written by third-party developers (highlighted red in figure 4.9), a number of precautions are already taken, such as isolating these on separate bridges, and having all source code undergo a rigid scrutineering process.

There are three core features of our system for resource description and authorization:

1. A root/top-level Hypercat catalogue is used to point to store-hosted catalogs to facilitate discovery and metadata retrieval by walking catalogs.
2. Distinct, granular macaroons are attached to any requests made to store endpoints that allow stores to independently determine whether a request is authorized or not.
3. Route-based permissions embedded in macaroons by the arbiter automatically control what operations an app or driver has been granted permission to execute, as long as the operation can be expressed as a route. This can be interfacing with data, or carrying out other functions explored below.

The relations between system components in this context are depicted in figure 4.9. The arbiter hosts a directory of all stores in the form of a Hypercat catalog. Our route-based permissions system applies to interfacing with Hypercat catalogs as well.

Macaroon caveats are conditions that must all be satisfied for the token to be considered valid. As such, using these would seem unsuitable for interleaving optional – or even mutually exclusive – permissions expressed as conditions, prima facie. This is however not the case, as individual caveats can be anything, including whitelists.

We therefore treat a set of caveats as a "product of sums", in the sense that each caveat is a conjunct that is ANDed with all other caveats, while individual caveats may define disjunctions in the form of whitelists with each contained disjunct ORed. Crucially, any combination of conditions can be expressed in this manner, making our extension of macaroons flexible enough to cover any expression of permissions.

The most critical caveat for our system of authorisation is one such conjunction: the *path* caveat of a route. We define a route simply as a combination of *target*, *path*, and *method*. Paths are strings, or string whitelists, with some optional formatting. They define accessible endpoints under a specified method for a single target. Methods are HTTP verbs that are generally mapped to CRUD operations, e.g. GET and POST to read from and write to a store respectively.

```
target = databox-mobile-store
method = GET
path = [
  "/cat",
  "/ws",
  "/profile/kv",
  "/accelerometer/ts/*",
  "/gps/ts/latest",
  "/logs/*/ts",
  "/(sub|unsub)/light/ts/*"
]
time < 1490790593391
```

Figure 4.10: An example of route caveats

Figure 4.10 is an example of a set of route caveats, to illustrate the flexibility of route-based permissions in the context of Databox APIs. In this case, we describe the extent of a bearer's permissions in a considerably detailed manner. For illustration purposes, these are all encoded into one token; in practice, we would separate the paths out one per token. For instance, the bearer of a token encoded with the above caveats can:

- Access the (potentially censored or filtered) second-level store catalogue of the target store

- Connect to a target store's WebSocket notification server
- Access a specific store-hosted JSON document with the key "profile" in a key-value database
- Access all "accelerometer" time-series endpoints
- See only the latest readings from store data source "gps"
- Read any logs as long as they are for time-series data
- (Un)subscribe to notifications for any new "light" data

Any operation that can be made into a REST-ful endpoint is automatically covered by this permissions system, without any extra configuration, making it exceedingly future-proof. As already alluded to, by separating routes, it is also directly compatible with existing APIs and can map directly onto them. Permissions can be made more granular by adding additional general caveats (such as an expiry timestamp) or endpoint-specific caveats (such as a bounds for time-series data, or a destination whitelist for the export service).

Authorization flow

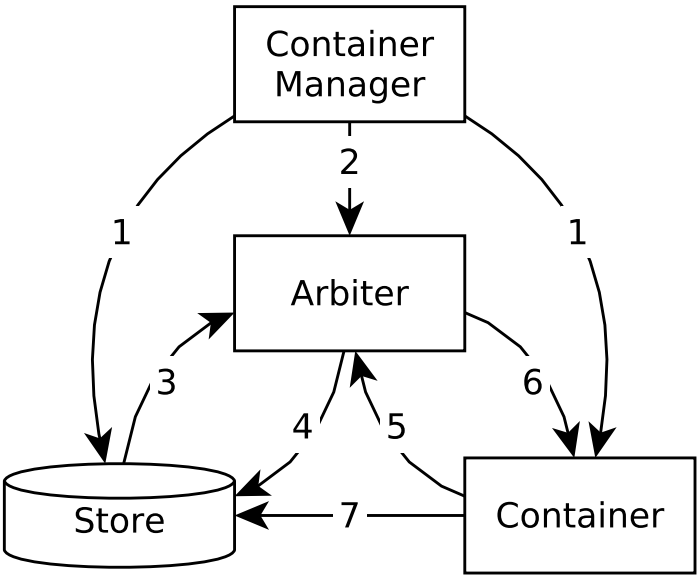


Figure 4.11: Overview of Authorization Flow

The authorization process can be enumerated as a series of seven steps, depicted in figure 4.11. These are as follows.

1. On launching a container (driver, app, or store), the container manager passes a unique token to it.
2. On launching the arbiter, the container manager passes its public key to it, so that the arbiter can verify signed updates. These updates comprise the extent of any given container's permissions.
3. A store uses its unique token to register itself with the arbiter.
4. The arbiter generates a shared secret key and responds to the store with it. The store can now verify bearer tokens minted by the arbiter and given to apps.
5. A container (driver or app) uses its unique token to request a bearer token from the arbiter. It may need to repeat this periodically as bearer tokens expire.
6. On checking the unique token against the permission records built in step 2, the arbiter generates a bearer token and responds to the container with it.
7. The driver or app container can now use this bearer token to write to or read from the store respectively.

Refinable bearer tokens

As the component that mints bearer tokens (the arbiter) and the components that verify them (the stores) are distinct, it becomes significantly more advantageous to securely encode permissions into the bearer tokens themselves, rather than checking them with the arbiter.

A standard way to do so already exists: Macaroons [67]; bearer tokens similar to signed cookies, to which one can add *caveats*. These caveats are conditions that must all be satisfied, and can be independently verified by stores using a secret key they share with the arbiter. Methodologies for translating permissions into caveats are discussed in section 4.3.2.

This is pertinent to implementation of authentication methods, as that largely depends on the low-level implementation of these components and the design constraints that this implementation may introduce.

The reason these are advantageous over kernel- or VMM-mediated capabilities, such as file descriptors, are mainly twofold. Firstly, in a cloud hybrid use case, their inherent cryptographic properties are exceedingly useful over the network, and do not require any additional layer of security. Secondly, these can be infinitely more descriptive than simple access control lists (ACLs) as the details of specific permissions can be encoded within them, stacked and nested, and independently verified by any store. This extensibility is further described in the following section.

Manifest permissions

As touched on briefly earlier, apps are packaged and downloaded alongside a JSON-formatted manifest file that includes app metadata, and more importantly, a list of permissions. These permissions are binary; either they are granted or they are not. Crucially however, a permission can be either *required* or *optional*. Thus, the subset of permissions that are required effectively describe a lower bound for all possible permutations of permissions, while both required and optional permissions describe an upper bound.

It is entirely possible to have multiple definitions of permissions pertaining to the same data source, for example with granularity requirements with a range of restrictiveness. Since a pair of permissions cannot be mutually exclusive, this essentially enables a user to "negotiate" the permissions they are prepared to grant to an app – in exchange for added features for example – from a UI that conveys risks, benefits, and possible inferences, as well as trust measures (ratings, reviews) on the app provider.

The result of this negotiation is a Service Level Agreement (SLA). The SLA is a list of permissions that will by definition include at minimum the required manifest permissions. This is the information that is passed on to the arbiter, and that is subsequently maintained by the arbiter so that it can mint bearer tokens.

Traditionally, such a token would be associated with a set of permissions of which any can be used at a given point in time. Unlike permissions however, caveats encoded into a bearer token must *all* be satisfied at the same time.

Macaroon caveats [67] can by design not be tampered with, as altering any information encoded within a macaroon would invalidate the signature. Only the minting party (the arbiter) and the verifying party (the store) posses the shared secret key to do so. It is possible however for the holder of a bearer token to add further caveats to it, and macaroons serve as a standard for doing so.

Some of these caveats are straightforward, such as matching the target container's name or preceding token's expiry timestamp, the formatting of which is illustrated below.

```
target = [name]
```

```
time < [timestamp]
```

Here a recipient container would reject the request if `name` does not match its hostname *or* if the time it received the request was greater than `timestamp`, indicating that the bearer token has expired.

To match this archetype, an SLA needs to be transformed once more in a context-

aware manner, such that types of permissions can be encoded into single caveats as whitelists.

The emergent capacity for negotiation, proves the transcription of static manifests into SLAs extensively advantageous over existing methods of managing permissions to access specific resources, such as access control lists. Meanwhile, the transcription of SLAs into caveats provides a robust and powerful mechanism to enforce these permissions securely and precisely on a network level.

Figure 4.12: Transcription of Permissions

Permission types

Data source permissions

The first and foremost type of permission, that can be specified in a manifest, is that regulating access to data sources. As already mentioned, there permissions are binary. Nonetheless, it is possible to refine these permissions on a per-source basis within a manifest.

For example, a user may through the dashboard agree with granting an app access to their Twitter timeline, but not their Twitter direct messages, although they come from the same source. An app developer may choose to make the former required but the latter optional. To do so, they would add two separate permissions to the manifest respectively, then as a property of these permissions, specify exactly which endpoints they request access to. This information is encoded by the arbiter into a caveat whitelist for independent verification by a driver. For example:

```
path = [
   ‘‘/statuses/home_timeline’’,
   ‘‘/direct_messages/*’’
]
```

Here, paths formatted in a way that can be compile to a regular expression [34] restrict what endpoints an app can access using the encapsulating bearer token. Crucially, the system is impartial to the context of these restrictions, as will be expanded upon shortly.

Concurrency permissions

So far permissions have been discussed in a "spatial" manner, i.e. the permissions that any given app has will be a subset of the permission space that is all sources at full fidelity. Another promising additional approach is to think of permissions in a "temporal" manner.

When it comes to potential inferences, there is some merit in doing so. Based off of priv-drop ideas from OpenBSD Pledge [105], we developed a method for drastically reducing the extent of potential inferences by automatically limiting what data sources can be accessed at the same time.

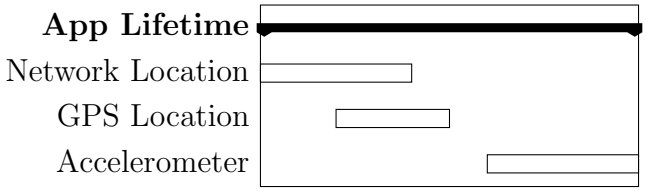


Figure 4.13: Example Data Access over an App's Lifetime

For example, an app that would access data in a pattern such as in figure 4.13, may never need to sample data from a user's mobile phone accelerometer and network location concurrently. Indeed doing so may permit additional unexpected inferences [47]. It might however need to access GPS sensor data to augment the coarse network location.

It therefore makes sense to disallow accessing multiple real-time data sources concurrently by default, and let app developers specify if they need to do so in a manifest. The duration of access is a separate matter (see *Granularity Restrictions*). Similarly, requesting laxer permissions over time can be collapsed to the single most lax level of permission. As such, this specification can be expressed as merely a set of lists of data sources that can be accessed simultaneously. The arbiter can ensure that an app never holds two valid tokens that can access conflicting data sources at the same time.

It is important to note that this does not however prevent drawing inferences from historical data over time. It is possible to mitigate this through granularity restrictions though – not to mention informing the user any credible risks of such inferences. Another possible point of extension would be to allow the definition of margins around accessing data, such that preventing an app from accessing accelerometer data within ten minutes of accessing GPS data would be possible for example.

All of this combined greatly simplifies and reduces the inference estimate that is presented to the user before launching an app, since one no longer needs to take account of every single data source used in relation to every other data source used by default.

Hardware and internet

The most unambiguous type of of manifest-defined permissions are those granting access to hardware resources (e.g. ephemeral memory and CPU) as well as devices physically connected through e.g. USB or Bluetooth.

More importantly however, apps may need to emit results of their processing or analytics to an external server in the outside world. As apps are completely isolated by default, this too must be specified within the manifest, especially so that it can be imparted to the user.

Granularity restrictions

As alluded to earlier, granularity restrictions are also specified in the manifest as properties of normal data source permissions. Granularity permissions are applicable to time series data, but can also be made to apply to any other kinds of data, since all data can be expressed as time series data, even if as one large data point.

Analogous to InferenceAide (formerly known as TussleOS) [111], granularity covers a number of parameters such as sampling rate (to limit the frequency of readings in a sample), batch size (to control the number of readings per sample), and periodicity (to throttle the frequency at which a data store can be queried).

There may of course also be data-type specific granularity requirements, such as the precision and accuracy of GPS coordinates, or the resolution of social media photos for instance. These can similarly be defined as properties of manifest permissions – provided the capacity to enforce them exists – and be transcribed into caveats.

4.3.3 Privacy-awareness through privacy contexts

Third-party *drivers* query external data sources and write data to system-managed *stores*. These are then queried by *apps* that perform analytics and, if necessary, only emit results back to third parties.

Figure 4.14 shows the components pertinent to this system. Here, solid arrows denote the paths that data can flow. As a single app can read from and write to many stores, these paths can manifest themselves as complex networks of cross-source analytics and derived stores. Our access control systems act at the red arrows. The *arbiter* mints signed bearer tokens with privacy thresholds embedded within them, and passes these on to apps and drivers. When interfacing with stores, these tokens are independently verified by said stores. Finally, any data leaving the box to be consumed by a third party

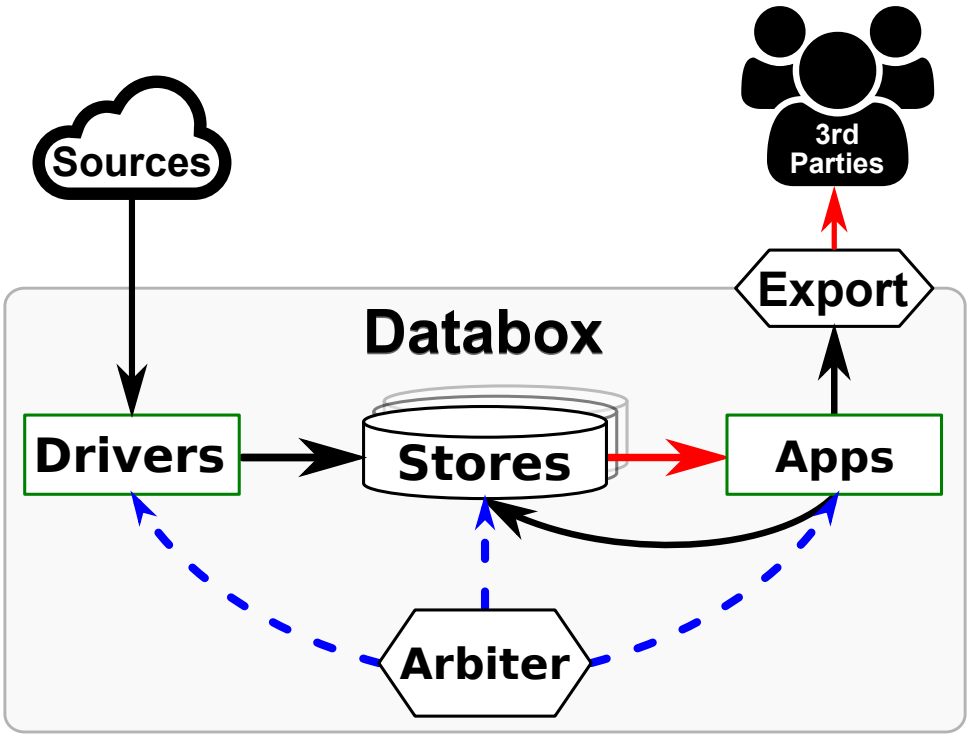


Figure 4.14: A high-level overview of Databox components

is similarly subjected to the same thresholds.

The core approach to how our system is deployed in this context is therefore twofold. As stores act as a border between producers and consumers, we first continuously update and maintain a *privacy context* for each data stream within each store using common privacy metrics. Then in tracking these metrics, we adjust the flows (in the simplest case by suppressing them or repeating old values) based on thresholds embedded within the tokens used to query a stream.

Our method can be adapted to any access control system that has a notion of per-consumer permissions, and as such, access control mechanisms are outside the scope of this paper. For our purpose, we implement a macaroon-based [67] bearer token system.

The *arbiter* in figure 4.14 mints tokens with embedded privacy thresholds (as macaroon caveats) that correspond to the permissions a bearer has. It maintains a record of these permissions that a user can modify at any time and take effect when an old token expires or is revoked.

The *arbiter* then cryptographically signs these tokens and passes them to data consumers potentially controlled by third parties (in this case *drivers* or *apps*). When a consumer makes a request to a *store*, it attaches the relevant token to the request. Permissions are embedded in these tokens, so the store/producer is aware of the privacy

context and permissions, and can verify these tokens through their signatures using a secret key shared by the arbiter and the store beforehand.

Thus, access control decisions based on privacy can be made on a request to request basis.

Marrying privacy metrics and access control

Our system functions between data producers and consumers. In Databox, this means between driver and app, between app and app, and between app and the outside world. Every driver and app must output data via stores, therefore the store is at first glance the most obvious place to implement our system.

There is however another alternative that is more versatile from a development perspective, which is to implement our system as a "privacy filter" app, that reads private data from one store, and writes transformed data to a derived store. This approach has the advantage of being modular as only a single type of store is needed, while any number of type-specific filtering apps can exist (e.g. an app that blurs faces in images). The increase in network traffic and added overhead would hurt latency however.

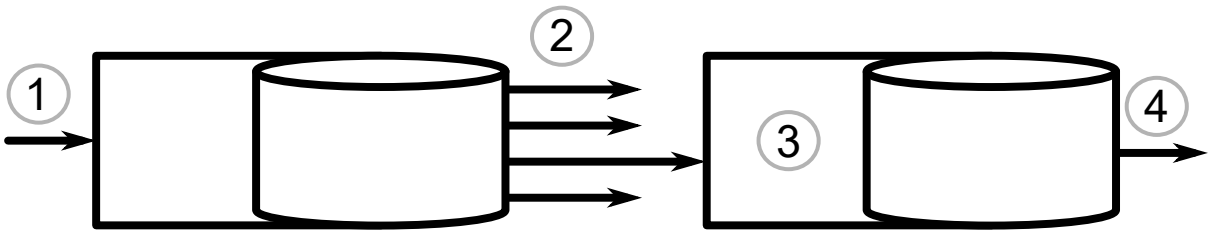


Figure 4.15: Stages of data transformation

We therefore implement our solution as an app, but future work may include hard-coding the most general types of time series aggregation into stores, and the monitors for the most common privacy metrics. Anything less common can be delegated to an intermediate app. Figure 4.15 shows this eventual pipeline — raw data enters at ①, is written into a store after being filtered/transformed at ②, and a second-order app ③ performs any additional more specific transformations before the data ④ is output in its final form.

4.3.4 Evaluation

Scalability

As a simple initial evaluation of of our system, a number of experiments were designed and conducted. The most pertinent of these tests are presented in this subsection.

While the multiplicity of drivers, stores, and apps may vary – i.e. one driver may write to many stores, one app may read from many stores, and one store may be read from by many apps – triplet units (see figure 4.16 approximate a typical modus operandi.

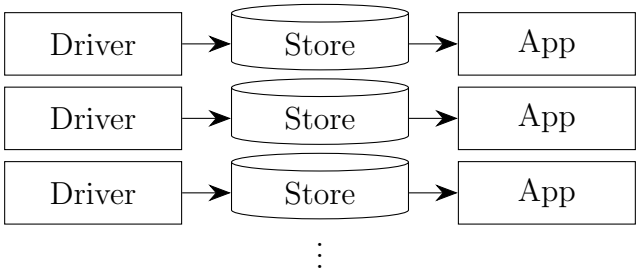


Figure 4.16: A Series of Container Triplets Launched

The purpose of this test is to show system viability, and that it is indeed feasible to scale up and run several dozen such units simultaneously, as would realistically run.

Then, as the system is strongly contingent on stores – read from and written to by apps and drivers respectively – these are focus of additional evaluation. For the first test, store throughput under maximum load while incrementally increasing the number of stores running simultaneously is investigated.

It is important to note that while throughput as a whole strongly depends on the database back-end used by a store, that is inconsequential to this experiment as only the change in throughput is relevant.

These experiments were ran on a range of database systems, some which had proven to be less optimized to this application; for example, an SQLite3 back-end (with no added logic for buffering queries) would need to run one query per transaction. While average hardware can process approximately 50k queries per second, it can only run about a dozen transactions, limited by hard drive speeds.

Procedure

A triplet unit is programmatically launched and – once the system has settled after launching the containers – statistics on every running container are subsequently collected through the Docker Remote API by sampling once per second over ten seconds. This

process is repeated for 0 to N units. Aside from the container types in a unit (driver, store, and app) statistics on the arbiter container are gathered as well. These statistics include the following.

- Percentage CPU usage by container type
- Memory usage by container type
- Sum network bytes transmitted
- Sum network bytes received
- Disk read and write (not pertinent to this particular test)

The experiment is ran on hardware analogous to what would run in the form factor of a home hub. To make certain that all hardware resources are only used by Databox processes, the experiment is ran on the smallest dedicated, fixed-performance AWS EC2 instance – specifically the general-purpose m4.large type. This runs on a 2.4 GHz Intel Xeon processor with 2 vCPUs and 8 GiB of memory.

The expected outcome of this test is to determine a general upper limit for how many containers can run simultaneously. As the number of triplets progressively increase, CPU and memory usages should be divided among all running containers, and the sum of network I/O for each container type should increase linearly.

To stress test these stores, a lightweight process is launched alongside them (see figure 4.17). After a short delay to allow the stores to initialize and settle, these processes proceed to bombard the store they were assigned to with write requests, and repeat on success ad infinitum.

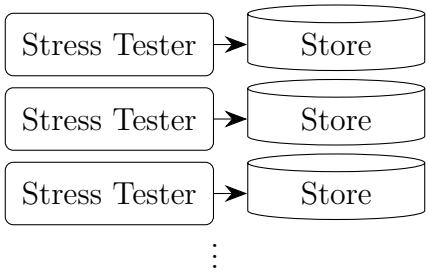


Figure 4.17: A Series of Store-Stress Tester Pairs Launched

At the same time, store operation must be evaluated with respect to the arbiter at the heart of the system. To do this, we look at the consequences of launching up to a hundred stores incrementally under normal conditions, and compare these to a control where the stores do not interact with the arbiter whatsoever.

While the first experiment demonstrates how well the system scales, the second exposes whether the arbiter is a bottleneck or not. We expect to measure evidence that the limiting factor is system memory, and that interfacing with the arbiter has minimal impact on system performance.

Results

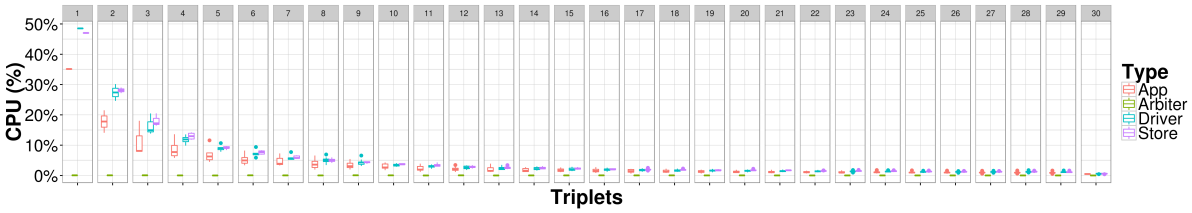


Figure 4.18: Percentage CPU Usage by Container Type

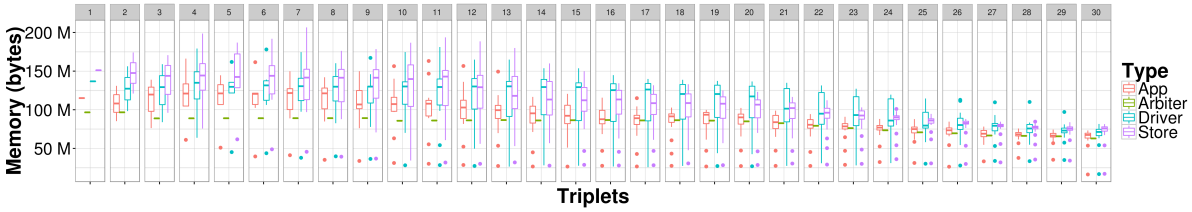


Figure 4.19: Memory Usage by Container Type

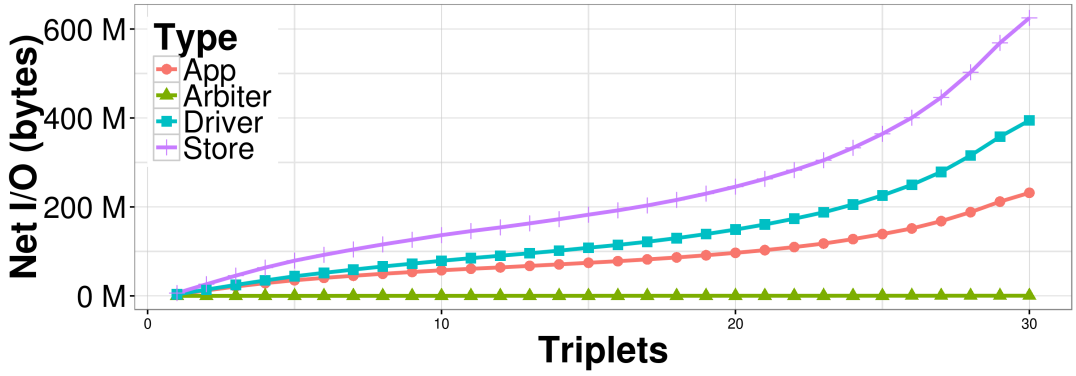


Figure 4.20: Sum Net I/O by Container Type

As expected, CPU and memory usage measurements diminish as the number of containers increase. Just under one hundred containers can run simultaneously without any issues (above ≈94). More interestingly however, is the sum of bytes transmitted and received by container type (see figure 4.20). Here, after launching twenty triplets, network I/O increases quadratically.

The results of the first store experiment (see figure 4.21) were as expected; the rate at which data is written to stores is not noticeably affected by the number of stores being

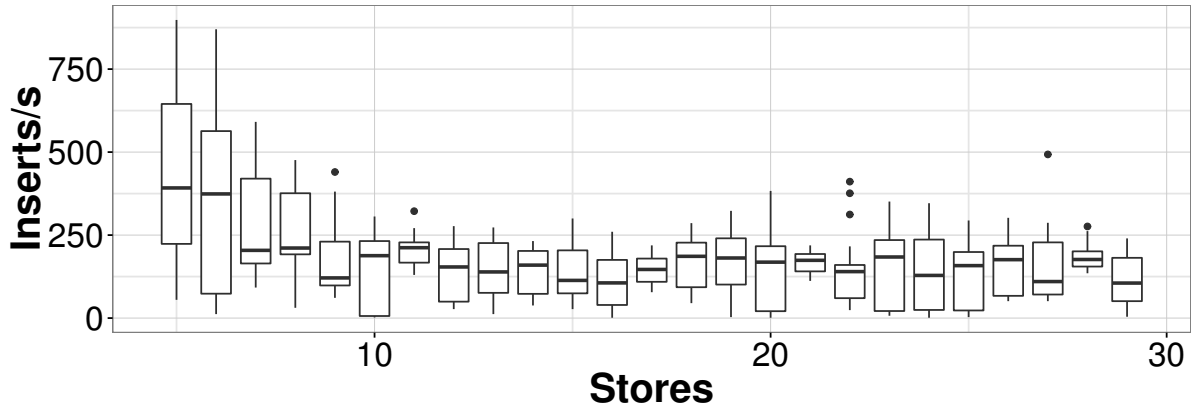


Figure 4.21: Inserts/s over Stores under Maximum Load

written to simultaneously. Indeed – as supplementary tests further indicate – when it comes to scaling up, the bottleneck is system memory.

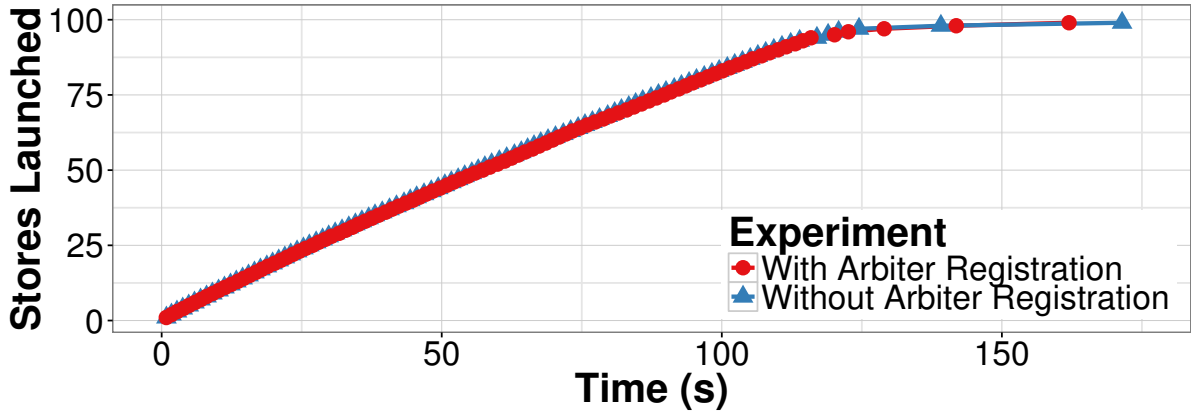


Figure 4.22: Stores Launched over Time

Furthermore, as the second store experiment demonstrates, The rate at which new stores are launched slows down as memory runs out at just below one hundred stores.

In addition, as can be gleaned from figure 4.22, the launch rates with and without arbiter interaction match almost exactly. The only discrepancy between the two is again indicative of the memory bottleneck, as the final containers with arbiter interaction take negligibly longer to launch. These results indicate that interfacing with the arbiter has no effect on the time it takes to launch additional stores successively.

Component overheads

The experiment described in this section seeks to answer the question of what the internal overheads of the Databox components are and how well they scale.

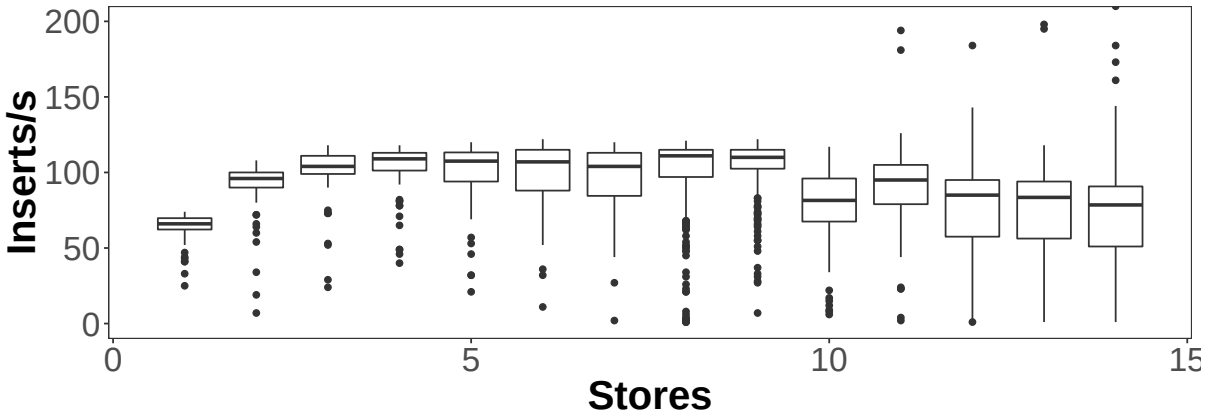


Figure 4.23: Aggregate *insert* throughput with extra stores. Tukey Box-Whisker plots: whiskers indicate 1.5× Inter-quartile range above (resp. below) the 3rd (resp. 1st) quartile. Remaining outlier points are plotted individually.

All experiments are carried out using a Raspberry Pi 3, model B, installed with the Raspbian Jessie Lite distribution and running Linux 4.4.50 with the swapfile size increased to 2 GB, except where indicated.

Databox software components are all constructed, distributed and deployed as Docker containers. As such, the primary overheads are image size and runtime memory consumption; CPU is not generally a concern, even on relatively constrained devices such as Raspberry Pis. The only component for which we must evaluate performance in terms of CPU utilization is the store, as this is on the datapath for all interactions between apps and drivers.

The image size of all components is negligible: most are between 70 MB and 90 MB each, except the Container Manager and the Export Service at 190 MB and 22 MB respectively. However, Docker images comprise several layers, which are shared where possible, so the aggregate on-disk size of all system components together is less than 200 MB. The runtime memory consumption of all system components stabilizes at around 53 MB due to the use of Node.JS. The exception is the Export Service, which is implemented in OCaml and uses just 8 MB.

Figure 4.23 shows results from stress testing stores as the key datapath component. We examine the aggregate *insert* (write) throughput achieved as the number of running stores increases, with all features (token verification, logging, etc.) in place. Each store is launched with a load generator client alongside that requests a token from the Arbiter, and then repeatedly inserts values to its store. All load generators are triggered simultaneously after stores have launched and initialized, and run for 60 s.

A single store achieves a median insert rate of 64.6 inserts/s, with the store running on

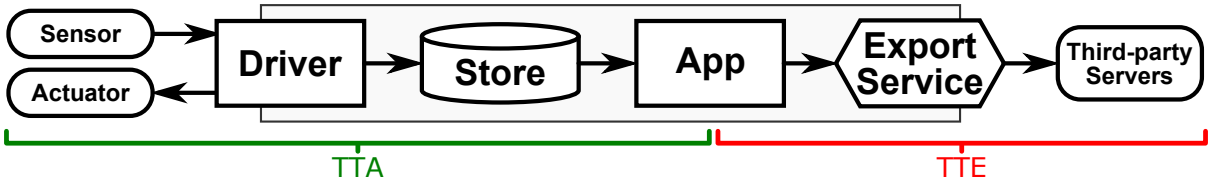


Figure 4.24: Sections of the data pipeline timed

one CPU core and the load generator on another. The host has four cores and so we reach a maximum median insert rate of 110 inserts/s with four stores. Beyond 10 stores, RAM is exhausted and the use of virtual memory affects performance: the very high outliers here are due to bursty servicing resulting in batching of insert handling. Memory constraints mean that no more than 14 stores can be launched. This demonstrates that the underlying platform components are sufficiently scalable on the sorts of small-scale, cheap hardware platforms we envisage deploying, such as home gateways. A fuller-featured but still small-scale platform, such as the Intel NUC, has more plentiful memory and so the number of stores that can be launched simultaneously more than doubles.

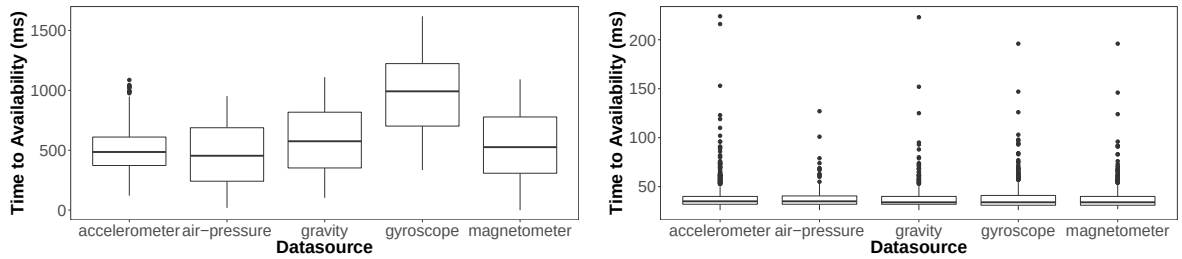
Our current implementation uses MongoDB [90] as, under these conditions, it performs better than SQLite3. This is due to the latter's transactional overheads slowing it down when logging every request/response, turning every read and write into full transactions, reducing the achieve rate from 50,000 queries/s to about 12 transactions/s.

External Overheads & Latency

This section examines what the external overheads imposed by the Databox are in terms of network traffic and particularly network latency.

Use of containers gives considerable flexibility in how Databox may be deployed: on a device in the home, on a virtual machine in the cloud, or a hybrid of the two. Figure 4.24 shows the standard simple data pipeline from single source directly to a single sink. We consider two measures: *Time to Availability* (TTA), the time between a datum being sensed and being made available in a store; and *Time to Export* (TTE), the time between an app initiating export of a datum and an external server receiving it.

In all cases, the Databox is started and an NTP server run on the same host. An Android smartphone runs an app that provides an API that uses SensingKit [64] to query all available sensors on the mobile phone. A companion driver runs on the Databox, configured to discover and connect to this phone. Timestamped sensor data is then streamed over this connection until the connection is severed by the driver. The data is formatted as uncompressed CSV with all entries having a timestamp ($\leq$17 bytes), and a



(a) Raspberry Pi (b) NUC

Figure 4.25: Time-to-Availability (TTA) for high-frequency sensors. Tukey Box-Whisker plots as in Figure 4.23.

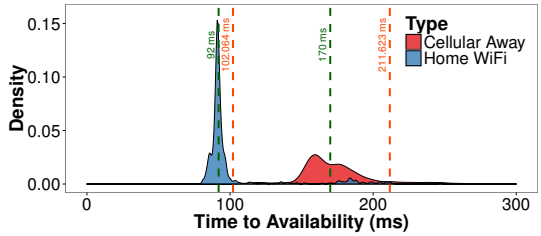
fixed number of sensor-specific columns/values. All entries have coordinate triples (x, y, z), except for air-pressure which has only a single reading (all $\leq$9 bytes each). Including separators, a single sensor transmits at most 4,800 bytes/s when sampling at 100 MHz, or about 415 MB/day. To avoid wasting bandwidth, the driver can toggle transmission of data from each sensor. We focus here on the high-frequency sensors rather than those that are low-frequency or transmit only when a change is detected (e.g., GPS, light-level).

The Time-to-Export behaves straightforwardly: it services different clients independently, ensuring one client doesn't get blocked behind another; time to export a datum is dominated by queuing delay in the service (if a particularly busy client) and the performance of the target external service (whether the target service performance itself, or the network connectivity to the target service).

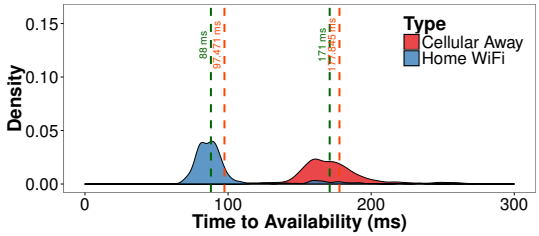
Time-to-Availability

In contrast to TTE, TTA is more interesting. Figure 4.25 examines the TTA for a given sensor of the many on the phone. While multiple sensors could be enabled simultaneously, this was limited due to eventual network contention between reporting sensors and the UI, as well as the driver bandwidth being limited due to the driver both receiving on the external interface and transmitting to its store through an internal virtual network interface. Figure 4.25a shows the median TTA on the Raspberry Pi to be around 500 ms, quite acceptable for most home automation and data processing tasks. For comparison, results for a somewhat more expensive device, the Intel NUC, are shown in Figure 4.25b, which reduces the median TTA to around 35 ms.

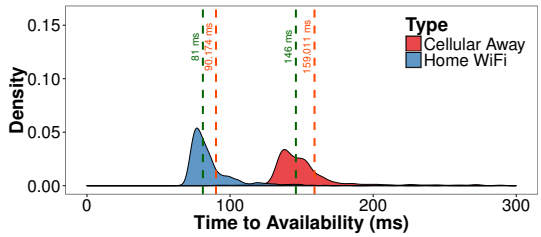
Figure 4.26 measures the TTA between the device and Databox for high-frequency accelerometer data, over both WiFi and cellular networks. Key points to note are: (*i*) the network type used to connected the device to the Databox dominates other effects for both mean and dispersion of the TTA; (*ii*) the variation between the different configurations



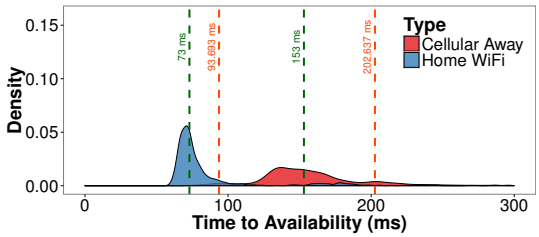
(a) TTA from device to in-home Databox directly



(b) TTA from device to in-home Databox via cloud VPN



(c) TTA from device to cloud Databox directly



(d) TTA from device to cloud Databox via in-home VPN

Figure 4.26: Measuring TTA between device and Databox over WiFi and cellular networks. Green and orange dashed lines show median and mean respectively.

is small and unlikely to be of concern for most users and applications; and (*iii*) the performance achieved by placing the Databox in the home with remote connections via a cloud-hosted VPN is not significantly different from any other organisation, but has benefits of privacy (placing Databox in the home ensures data remains under the user's direct control) and reachability (placing the VPN server in the cloud ensures it is easily reachable when the device is roaming outside the home).

In more detail, Figure 4.26a gives the baseline where the device and the Databox are co-located in the home. As expected, this has the minimum dispersion for both networks, but particularly Wi-Fi. Somewhat surprisingly, Figure 4.26d where the device connects to the Databox using a VPN hosted on the Databox, has slightly lower mean TTA, though higher dispersion. This is however within the NTP error margin. Finally, Figure 4.26b, perhaps the most realistic deployment scenario where the device connects via a cloud-hosted VPN to the in-home Databox, has slightly higher and more dispersed TTA though differences are sufficiently small that it is unlikely a user would notice or an app be significantly affected.

Privacy-awareness performance

While §3.3.1 showed that significant privacy gains can be made without degrading utility, previous work has achieved more impressive privacy/utility trade-offs [73]. Where our work differs is that it is also efficient enough to run online for real-time streaming data on cheap hardware.

In this section we show this by implementing our system over Databox, running it on typical hardware (an Intel NUC6i3SYH), and measuring the added latency in the pipeline. We examine the difference in *time to availability* (TTA) — the time between when a sensor emits a sample and when it becomes available in its final form to an app at the end of the pipeline. We can of course repeat the measurements on all derived data ad infinitum, but this has limited practical benefits.

We measure this latency for 20k samples under three conditions:

- *Datastream*: Vanilla Databox times as a baseline. Access control is binary and at the datastream level.
- *Surprisal*: Inclusion of our system in the pipeline using surprisal as a privacy metric with a fixed threshold and infinite window size.
- *Windowed Surprisal*: The same as the previous experiment, but with a falloff of one minute.

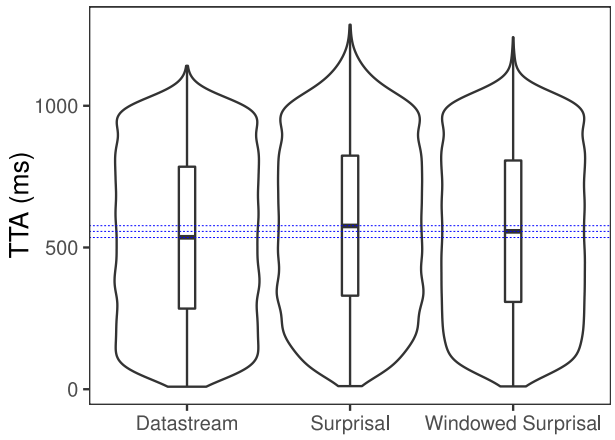


Figure 4.27: Distributions of time to availability under different conditions

Figure 4.27 shows the density of these latencies. The means, with blue dotted lines drawn through, are at 535.1633 ms, 576.9499 ms, and 556.7980 ms respectively. The difference between with and without surprisal calculation is negligible and well within

the tolerance for real-time applications. The added latency is small enough that privacy filtering at this fidelity is possible without impacting user experience.

The small difference of 41.7866 ms is only because the calculation gradually gets slower as the number of samples increase. This can be mitigated by limiting the number of past samples processed (in this case the window is one minute or 6k samples long). As soon as the window is saturated, the upwards trend in latency flattens and remains constant. This way, the difference in latency was further reduced by almost half to 21.6347 ms.

4.4 Summary and discussion

Our scalability evaluation results show that it is feasible to scale up utilisation without significant overhead on a PC similar to home-hub hardware. Additionally, the first experiment shows that a central arbiter is able to deal with authentication and queries from thousands of apps and queries, while enabling flow-monitoring and auditing tools to monitor the transactions performed using access tokens.

The network profiling test shows that after launching twenty triplets, network I/O increases quadratically (see figure 4.20). This is due to packet retransmission, and does eventually lead to congestive collapse.

Currently, the network is partitioned in such a way that all drivers are connected to one bridge, all apps to another, and stores and the arbiter to both. The rationale behind this is that apps and drivers would never need to directly communicate and so do not need to be able to to begin with.

One potential way to raise the number of containers that need to be running concurrently for congestion to affect performance is to instead partition the network horizontally, as opposed to by type vertically. Only containers that need to communicate with each other should be able to (in this case the triplets) and no driver or app ever needs to communicate to any other driver or app. This also has added security implications as it would make cross-container data leakage impossible on a network level.

Our remaining evaluations show how the latencies we incur every step of the way are more than made up for by the advantages our architecture provides (running computations at the edge). Similarly, our performance evaluation of the privacy-awareness augmentation to our system shows that additional latencies are negligible making our system feasible on all levels.

Chapter 5

Edge computing in the browser

5.1 Overview

So far, we have examined distributed edge computing in the cloud and home IoT devices. This chapter goes a step further and looks at processing in the browser for applications that are specifically intended for that environment.

We begin by examining the research landscape in this area as it stands and introduce all pertinent concepts and vocabulary. We then present our contributions to this area and the implementation of our approach as well as other existing approaches. Finally, we present a framework for evaluating our system in different contexts and evaluate it against others, showing where ours is superior.

5.1.1 The web ecosystem

In §2.2, we have shown the trend towards microservice architectures and greater granularity. Serverless computing has been the clearest recent example of this. Many application providers still rely on older technology however and are locked in to legacy architecture due to high technical debt. The transition to cheaper and more efficient architectures is therefore slow.

As it stands, the standard approach for deploying a web app is to rent a Virtual Private Server (VPS), often with one of the big cloud providers such as Amazon Web Services (AWS). Most of these have tools for handling scaling and load balancing out of the box and host from big datacenters in well served locations.

In the very early days of the internet, when IPv4 addresses were thought to never run

out, and NATs were not needed, webpages were being served from the same machines that request them. A quarter of a decade ago, the web was largely made up of static pages. With the advent of PHP and similar, server-side processing spawned a large range of new web app functionality. To this day, some degree of server-side processing is all but expected. PHP continues to dominate at 79% use by all websites with a known server-side programming language, while static files sit at 2.1% [120].

Today, large scale web application providers employ a multitude of optimisations in order to keep their service performant. The use of Content Delivery Networks (CDNs) is an example of one such optimisations, where web application providers may deploy or rent caching proxy servers that are geographically closer to their users in order to deliver resources such as static assets to them quicker. Some web application providers, especially those active in areas with low latency requirements such as real-time multiplayer games, go as far as operating private networks [85, 86, 87].

security, other trends

5.1.2 Beyond serverless

The value in browser dominance has been widely recognised. As of today, Google Chrome continues to hold the majority market share on the battlefield of the browser wars [113, 112]. At the same time, browsers (at least the one adhering to modern standards) are becoming more and more capable.

It has gotten to the point where a browser is able to replace most, if not all, equivalent desktop applications. Google's Chrome OS goes as far as extending the Chrome browser to become an operating system in itself, with web apps replacing desktop apps.

With users spending the majority of their computing time in the browser, this is not a surprising development, and with browsers building cross-platform APIs for applications that require lower-level hardware access (such as WebGL, WebVR, and recently, WebUSB), soon there will be very little that cannot be done within the framework of a browser. Indeed, this gives developers a different way of thinking about software development that is more dynamic, standards-drive, and OS-agnostic.

From the other side, web development has also changed to adapt to this shifting environment. With most users worldwide browsing the web through smartphones [114], especially in developing countries where a smartphone is a much smaller investment for connectivity, web developers have begun to take a "mobile-first" approach in their design. The standards have shifted to reflect this too, through media queries and similar.

Meanwhile, as devices have been getting more powerful, software design patterns have changed to reflect that. SPAs, and more generally, CSR has become prevalent and standard, limiting the back-end to REST APIs and minimising interaction with servers. Web frameworks that enable this, such as React, Angluar, Vue.js have soared in popularity [61] and front-end employers now usually expect familiarity with one or more of these frameworks.

This also means a greater emphasis on client-side security, the brunt of which is handled by web standards and, by extension, browser vendors. APIs such as Content Security Policy (CSP) are available to developers to prevent Cross-Site Scripting (XSS) and attacks in the same class or similar.

Web apps have also infiltrated the desktop space. This was originally through frameworks such NW.js (previously node-webkit) and Electron, but has now become a standard with PWAs. PWAs also allow web apps to become mobile apps, a feat which was previously only possible by wrapping your web app in a native WebView object, or porting a web app to an equivalent of that with software such as Cordova or PhoneGap. PWAs introduce a whole new way of thinking about and writing web apps, that employ service workers to cache resources and enable apps to be usable offline too. Popular apps are already secretly web apps, and it is not inconceivable that the majority of desktop and mobile apps will be the same in the future.

In previous chapters we have explored the serverless paradigm, where the trend in cloud computing is tending towards smaller, more granular microservices with shared overheads. This trend has strong parallels at the client-side. The aforementioned shrinkage in server-side processing, and bare REST APIs that map closely to Create, Read, Update, and Delete (CRUD) operations on databases, go hand in hand with the trends we have discussed in this section.

The critical issue here is that while the back-end has been getting leaner, the front-end has seen a lot of bloat and frustration by developers — the rate at which new front-end JavaScript frameworks have been spawning is impossible for most to keep up with and causing a lot of fragmentation. Meanwhile, the very concept of SPAs intuitively does not mesh well with sane design practices employed when creating microservices, such as the *Separation of Concerns* or the *Single Responsibility Principle*.

5.1.3 Modern use cases

The different web app paradigms we touched on in the previous section are all worthy of detailed research in themselves, and depend very much on the application in question.

For example, social media platforms may need their different functionalities split up in different ways where standard CSR and REST simply are not enough. These requirements literally drive the development of the currently most popular front-end framework, Facebook's React, and others like it.

Meanwhile, online media-services providers like Netflix, that serve a large volume of static data, instead optimise through compression and replication. In these cases, infrastructure is more important than page load times for example.

Minimising latency is however becoming a much more prominent focus than throughput. Aside established areas like trading stocks, we are seeing emerging technology with critical low latency requirements, such as Virtual Reality (VR) and IoT. To meet these requirements are solutions at the protocol-level (e.g. HTTP/2) as well as infrastructure-level (e.g. 5G and satellite constellation networks).

The current state-of-the-art in serverless computing are Cloudflare Workers [11] which, in addition to runtime optimisations, run on servers as close to the user as possible. A common theme in our work has been the utility of pushing computation even further towards the edge — to the actual devices of the users.

In this chapter, we look at one very specific use case and examine the advantages and disadvantages of doing this. This use case is that of interactive applications, such as multiplayer games. We focus on this area because it is an extreme case that amplifies all the challenges with other use cases. This is because:

- Latency and even throughput requirements are very high compared to other use cases
- Server-side logic is usually monolithic and complicated (e.g. physics simulations)
- Because of this, costs are much more significant and are difficult to cut down through porting to e.g. a serverless architecture. This increases the developer barrier to entry
- There are strong incentives for bad behaviour (e.g. cheating) so client-side processing cannot be trusted
- Interaction between users can be innumerable and simultaneous
- For some sub-types of these applications (e.g. Massively Multiplayer Online Games (MMOGs)) solutions that are simple, scalable, low-cost, and low-latency are virtually non-existent

This area is especially challenging because the development of the architecture of the internet was not driven with this use case in mind. It is therefore important to drop all assumptions from the start and consider different network architectures and systems when searching for a solution.

5.1.4 Peer-to-peer versus client-server

The web browser as a target platform for modern games has been unpopular due to its limitations, for example lack of multithreading support. In recent times, developers have been deploying less and less to the browser [38, 39]. However, as modern HTML5 APIs such as WebVR, WebGL, and WebAssembly are beginning to see widespread support in browsers and devices, there is an opportunity for independent game developers to rediscover the browser as a serious target for deployment. As independent developers are more limited when it comes to labour and resources, many shy away from large-scale, real-time multiplayer game development.

The most common approach to supporting multiplayer is a client-server model, where one or more authoritative servers maintain communication between clients. This holds true for both browser and non-browser games. This is usually (at least at first) the least complex solution and fits well within existing internet architecture and paradigms.

The potential of P2P architectures for game networking has always been recognised however. These have the capacity to lower costs for game developers and provide a more reliable service to the players. In the past, there have been many non-browser games that employ peer-to-peer (P2P) communication. As most clients/peers tend to be behind NATs, and port-forwarding is often a non-trivial ask of players, these games rely on techniques such as UDP hole punching, or supplement communication with intermediate servers. In most use cases, only a small group of players need to be connected, and one peer in a group of peers is designated "host" and acts as a de facto authoritative server. This limits the server costs of the game provider to simply acting as a lobby/directory for finding these rooms/groups, but at the same time, the number of players that a host can support is more limited than a standalone server. This is a critical limitation when it comes to implementing the same for games with many players that have been recently becoming more popular, such as the Battle Royale or MMO genres, which we aim to address.

5.1.5 Related work

When referring to virtual environments with many interacting users, such as MMOGs, the literature uses scattered terminology. In this work, we use the all-encompassing term Virtual Environment (VE). Table 5.1 lists other terminology used in the literature.

Table 5.1: Acronyms in the VE space

Acronym	**Expansion**
VE	Virtual Environment
DVE	Distributed Virtual Environment
NVE	Networked Virtual Environment
CVE	Collaborative Virtual Environment
MMVE	Massively MultiUser Virtual Environment
SVW	Social Virtual World
DIA	Distributed Interactive Application
DIS	Distributed Interactive Simulation
MMG	Massively Multiplayer Game
MMOG	Massively Multiplayer Online Game

Within this field of research, there are also a number of commonly used terms and acronyms. Here, we list the most common of these.

The area around an avatar in a virtual environment is commonly called the Area-of-Interest (AOI) [109, 71] in contemporary literature. In earlier work this has also been referred to as an *awareness area* [68], *aura* [43], *Domain of Interest* [92], and *aura nimbus* [7]. The act of limiting the information that individual players have access to based on their AOI, for reasons of increasing scalability and/or decreasing cheating, is referred to as Interest Management (IM) [7, 92].

IM is generally divided into *spatial* techniques where players move across a continuous world and IM is computed dynamically, and *regional* techniques where the VE can be split into discrete zones that players can move between. In this chapter, we focus primarily on spatial techniques.

Existing work has also been categorised into *structured* and *unstructured* P2P [124, 40]. Generally, structured approaches utilise DHTs or similar distributed data structures, while unstructured approaches connect peers based on factors such as VE positions. We focus on unstructured systems as our contributions lie in using factors such as these to build more optimal topologies.

Systems such as Kiwano [27], VoroGame [16], and [75] seek to solve issues of scalability

for MMOs. These systems utilise many of the methods we explore, however they are either cloud-based or a hybrid of cloud and P2P. We seek to remove dependency on the cloud as much as possible in order to hosting costs.

Solipsis [68, 36] was one of the first approaches at pure P2P architectures for virtual environments. This was taken further through VON [55] and subsequent work [56]. These approaches use virtual avatar position data to compute the peer network topology based on area of interests. Most methods use Gnutella-like initialisation through distributed hash tables (DHTs) or similar. Newer methods account for variable spatial concentration of players by using Voronoi-based or derivative partitioning schemes. There have been several extensive survey papers covering these methods [124, 40].

While our proposed method builds on a lot of this past work, we take it further by allowing arbitrary metrics for computing ideal topologies. In addition to this, we guarantee that our peer networks are fully connected and resilient to a certain extent of link failures. Further, we consider the challenges of implementation within the browser; past methods have been platform-agnostic.

TODO: Add papers from spreadsheet

5.2 Cheap, scalable distributed virtual environments

5.2.1 Design

We have established that most existing solutions generally take the same approach — completely connecting peers in a certain virtual AOI. This means that the peer network topology is dependent on how these AOIs are defined, which in turn is usually dependent on virtual peer positions.

We approach this problem from a different, general angle. In this section, we begin by defining the problem we solve more explicitly. Then we formally delineate the details of our solution while justifying our design decisions, and how our method can be extended, throughout.

Definitions

Given a set of peers and information on these peers, we aim to design an algorithm for building a peer network topology that is *fast*, *resilient*, *efficient*, and *extensible*.

Fast — we characterise speed through a distribution of pairwise latency. A faster

network will not only have a lower mean latency, but also a lower upper bound. The upper bound is equivalent to how fast a single broadcast message can propagate through the entire network.

Resilient — we define resilience of an network by its connectedness, specifically *k-vertex-connectedness* and *k-edge-connectedness*.

Efficient — a network is more efficient than another if it is more sparse; when there is a fewer number of total edges, ceteris paribus.

Extensible — we optimise our computed topologies based on a number of default factors, which we enumerate in this section. Our system is extensible if these factors can be augmented with others.

Algorithm

In §5.1.5 we established how most existing solutions only look at VE user positions for IM and therefore how the peer network topology is computed. Our solution also considers positions, however these positions are not raw VE positions, but rather n-dimensional coordinates that are computed based on a variety of default factors, which we call distance metrics. These are:

- VE user positions
- Pairwise latency
- A pairwise trust score
- Time connected

As we focus on distributing the relaying of states, we relegate peer network correlation to a centralised server. The data transmitted to do this is comparatively low in volume, however our architecture can be fully distributed in the future, in a similar fashion to Vivaldi [25]. In current architecture, the above metrics are collected by a coordinating server which then computes a topology and instructs peers on which other peers to connect to or disconnect from.

The above metrics can be extended to include arbitrary factors, however this has some limitations. The first limitation is that these metrics have to satisfy the triangle inequality. This is often assumed, for example with Network Coordinate Systemss (NCSs) such as Vivaldi [25], although practically this may not be true, simply because of sub-optimal routing on the internet and heterogeneous routes. For example, with nodes A, B,

and C fully connected, the path from A to C may be faster through B than a direct path. We investigate how we can mitigate this issue through increasing the dimensionality of the final coordinates.

The second limitation is that the metrics have to be pairwise/undirected. A simple distance measure is the same both ways, however a trust score is subjective (and again may violate the triangle inequality). In order to overcome this limitation, we combine directed metrics by using their mean, max, or use a different statistical average depending on the metric. These two limitations also enable important optimisations which we discuss later.

Figure 5.1 illustrates the steps of our method from the collation of distance metrics to instructing peers to update their connections. In this section, we discuss each step in this pipeline in detail.

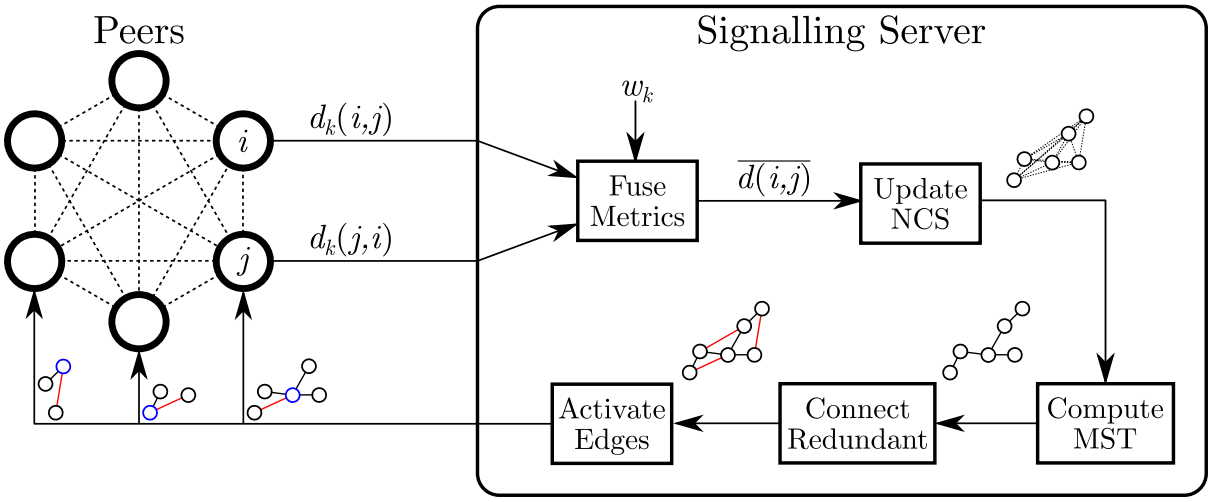


Figure 5.1: Our peer network topology computation pipeline

Distance metrics

Each of the above four factors are computed on a per-edge basis. To capture the first factor, we calculate the euclidean distance between the VE positions of nodes i and j, which we call x_i and x_j respectively. These coordinates can have an arbitrary dimensionality, M (usually 2 or 3), so we denote the dimension as m, i.e. $x_i = (x_{i1}, x_{i2}, ..., x_{iM})$. We refer to this variable as $d_p(i,j)$ defined in equation 5.1.

$$d_p(i,j) = \sqrt{\sum_{m=1}^{M} (x_{im} - x_{jm})^2} \tag{5.1}$$

To compute the latency factor, we simply divide the round-trip time between a pair of nodes, rtt_{ij} for nodes i and j, by two. This does assume that the latency one way is the same as the latency back, which is not necessarily true practically, but this is common assumption that is made in related work [25]. We refer to this variable as $d_l(i, j)$ defined in equation 5.2.

$$d_l(i, j) = rtt_{ij}/2 \tag{5.2}$$

Each node i maintains a trust score for every other j denoted as t_{ij}, where $0 \leq t_{ij} \leq 1$. t_{ij} is better thought of as as *distrust* score, as we invert it here to save the step of inverting it later (since more trust is analogous to a smaller distance between peers). We initialise this directed trust score at 0, trusting peers by default, and it changes dynamically over time.

Trust is computed automatically based on how frequent a peer sends an incorrect state. As peers have no way of telling if another peer is behaving maliciously or simply have a corrupted game state, we compare state to that sent by other peers and mark the majority as "correct". If a peer continues to bad state, then they become less and less likely to be kept as a connection, or have new connections initiated with them. This can be overridden by a peer manually marking another peer as untrustworthy (e.g. through higher-level observations such as game-layer cheating) which can set this factor to 1 or an extremely high number, effectively associating a huge cost to connecting to this peer.

Our system is versatile enough to allow other trust/banning mechanisms to be encoded within it in a similar fashion (such as sharing trust scores with other trusted peers), however the trust metric we describe serves as a basic proof-of-concept that encoding application-critical, directed factors into the topology computation decision can be done.

Note that t_{ij} is not necessarily equal to t_{ij}, making it a directed factor. To transform it into a pairwise/undirected factor, higher distrust takes precedence. This makes sense intuitively, as it allows peers to override malicious peers who try to artificially paint themselves as more trustworthy. We call this transformed variable $d_t(i, j)$ defined in equation 5.3.

$$d_t(i, j) = \max\{t_{ij}, t_{ji}\} \tag{5.3}$$

Finally, a common problem that has plagued previous work is the frequent switching of connections. Practically, there is a cost associated with connecting to a new peer, as the initial handshake adds overhead latency. Some approaches have tried to minimise

this overhead by attempting to create topologies that are much more stable over time as peer positions change [15]. There has even been work that characterises the way that players move in a VE in order to anticipate how this motion will affect churn, and this information can be used to augment how the network topology is computed [58, 78].

While both of these areas are important, the first can be applied to any system that uses Delaunay triangulation, so it is not an alternative to our system but a potential future avenue of optimisation, and the second is very application-specific.

We instead use concepts from the load balancing space to solve this problem in a simple, general way. It is not uncommon for load balancing systems to adopt a notion of "stickiness" when routing, in order to add some temporal consistency to links and prevent sporadic switching between two similar queues or sinks. We discussed this idea in more detail in chapter 3.

We therefore capture the time a connection between two peers has been active, s_{ij}, as an additional factor when computing our network topology. This is a straightforward pairwise measure (identical to s_{ji}), however its effects should not be linear, as it makes more sense to have a kind of exponential backoff where new connections are more likely to stay and older ones are more equivalent.

As such, we use a simple sigmoid function, the hyperbolic tangent, over the time connected, so that it can be treated as a linear factor in the next step of the process. This can be adjusted to make the connections more or less "sticky", so we include the constant c_s to denote this. This constant controls how fast a connection converges in stickiness to other older connections, *not* how much the stickiness variable as a whole affects the topology computation. A lower constant means that this will happen faster (less sticky).

We also reiterate that these factors can be the result of arbitrary computation that should affect the topology in some way. We call this "stickiness" variable $d_s(i, j)$ defined in equation 5.4.

$$d_s(i, j) = \tanh\left(\frac{s_{ij}}{c_s}\right) \tag{5.4}$$

Finally, we define the full set of pairwise distance metrics as $d(i, j)$, which for our implementation is made up of $\{d_p(i, j), d_l(i, j), d_t(i, j), d_s(i, j)\}$, but can be extended arbitrarily. For all these metrics, where $i == j$, the value is 0, and the pairwise/undirected limitation will mean that swapping i and j will yield the same value (e.g. $d_t(i, j) == d_t(j, i)$).

Our Network Coordinate System

To build our coordinate system, we make use of well-understood concepts. Given a graph with weighted edges, we can simulate a spring system that computationally/iteratively moves nodes to optimal positions with minimised energy. This is often used in data visualisation when rendering force-directed graphs — Hooke's law (which describes the behaviour of springs) is applied to edges and Coulomb's law (which describes the behaviour of charged particles) is applied to nodes.

This idea has been used before in the context of NCSs, by the distributed NCS Vivaldi [25]. Among NCSs, Vivaldi [25] is indubitably the most widely used, primarily because it is very easy to implement, can be distributed, and gives good results. Other NCSs solve some of the limitations of Vivaldi at the cost of complexity. For example, Pharos [20] has better accuracy than Vivaldi by using two separate overlays with two separate sets of coordinates for long and short link predictions. Meanwhile, Phoenix [21] solves triangle inequality violations by using a matrix factorisation model.

Our model is similar to Vivaldi, with some important differences. Firstly, we take more than just round-trip times into account in an adjustable manner. Secondly, we do not limit the computed coordinates to just three dimensions. We also show that doing this can mitigate triangle inequality violations in a much simpler way. Finding the optimal number of dimensions to use is a future goal of our work.

The first step is to normalise and combine the metrics we have described into a single metric that we can use as spring parameters. To do this, we apply a weighted average. Each weight effectively dictates how much influence single metric has on the positioning of a node within our NCS. These weights are referred to as w_k in equation 5.5 and map directly to $d_k(i,j)$ for K weights/metrics. These weights are normalised. The final combined metric, the edge cost, we call $\overline{d(i,j)}$.

$$\begin{aligned} \overline{d(i,j)} &= \sum_{k=1}^{K} w_k d_k(i,j) \\ \text{where} \sum_{k=1}^{K} w_k &= 1 \end{aligned} \tag{5.5}$$

From this point onwards, our method is identical to Vivaldi except for three important differences:

- Vivaldi's L_{ij} is substituted with our $d(i,j)$
- Vivaldi simulations show that 2-dimensional coordinates provide adequate accuracy

for their use case, while our use case requires coordinates with a higher dimensionality

- Our method favours larger values of t (a constant in Vivaldi that controls the step size for moving nodes). This is because our collection of metrics has the propensity to change much more rapidly, meaning the network coordinates need to reflect that much more aggressively. The downside is that the network positions can be more turbulent.

It is also important that this coordinate system is euclidean. Previous work has shown the advantages of using non-euclidean models, however this would make our system very difficult to scale, as we rely on this property when computing spanning trees in the next step. Euclidean Minimum Spanning Trees (EMSTs) allow optimisations in calculation that brings our method to a level of efficiency that is practical.

Computing topologies

The final step in our process is computing a topology that fits our constraints over the nodes with the computed coordinates. This is an NP-hard problem, and an optimal solution can be used to compute Hamiltonian paths. Our solution is a heuristic which we later compare to optimal solutions found through brute force.

So far, we have shown how our solution is *extensible*, and we evaluate how *fast* it is experimentally. Our computed topologies must additionally satisfy the criteria to be *resilient* and *efficient* that we define in section 5.2.1.

In order for these topologies to be efficient, we start by computing the provably most efficient configuration: the Minimum Spanning Tree (MST). Here, our edge costs correspond to $\overline{d(i,j)}$. Over time, we compute the edge costs for every pair of distinct nodes and build a complete graph of edge costs. When we physically connect two nodes in the network, we say that the edge has been *activated*.

An MST creates many bridges however, which makes our network very fragile; a single node disconnecting or a single link failing can disconnect the network graph. To remedy this we add connections using algorithm 1. Here, `peers` is a list of peers, each containing a list of edges, each associated with a cost $\overline{d(i,j)}$.

The result is an MST network with added connections to bring the k-edge-connectivity of the network above a certain predefined threshold. This is not optimal, but we compare the resulting network to bruteforced optimal topologies.

```
Data: peers
Input: minK, the minimum edge-connectedness
Result: A network that is efficient and resilient
Compute the MST using Prim's algorithm and activate MST edges;
foreach peer of peers do
    /* Ignore adequately connected nodes                              */
    if peer.activeEdgeCount < this.minK then
        /* Half cost of edges connecting 2 weakly-connected peers     */
        foreach edge of peer.edges do
            if edge.peerA.activeEdgeCount < minK and
              edge.peerB.activeEdgeCount < minK then
                edge.modifiedCost = 0.5 * edge.cost;
            else
                edge.modifiedCost = edge.cost;
            end
        end
        Sort peer.edges by modified cost;
        /* Activate cheapest edges to adequately connect this peer    */
        foreach edge of peer.edges do
            if not edge.isActive then
                activate edge;
                if peer.activeEdgeCount ≥ minK then
                    break;
                end
            end
        end
    end
end
```

Algorithm 1: Our heuristic algorithm for adding resilience to peer networks

We modify half the cost of edges that connect two under-provisioned peers, as activating these kills two birds with one stone by adding an edge to two nodes at once. We also recognise that a minimum k-vertex-connectivity may be a constraint that more closely maps to resilience, and we intend to include this in our algorithm in the future.

Topologies

Our method described above computes topologies that are well suited to large-scale VEs such as MMOGs. We recognise however, that different applications, and even genres of games, may have different requirements. We therefore made our library general-purpose, implementing different kinds of methods for computing topologies based on a developer's requirements. We also implemented APIs for allowing developers to build in their own algorithms for computing topologies, as well as evaluating them. This also allows us to

easily compare these different methods, which we do in section 5.2.3.

Table 5.2 lists the topologies our framework currently implements, which scale of VE they are most suited to, and the parameters required. The scale column serves to illustrate how many players in general a given topology would be able to support. Small-scale topologies are for environments with discrete "rooms" that can support a number of players usually in the single digits. Examples are collaborative editing or simple First-Person Shooters (FPSs). Large-scale topologies can support a theoretically infinite number of players and are well suited to continuous MMOGs and location-based Augmented Reality (AR) games. Medium-scale topologies sit somewhere in the middle as they mitigate a lot of the scalability issues of small-scale topologies. These can potentially be suited for Real-Time Strategys (rtss), Multiplayer Online Battle Arenas (MOBAs), or Battle Royale games.

Table 5.2: Supported P2P Topologies

Topology	**Scale**	**Parameters**
Complete	Small	
Host peer	Small	
Superpeers	Medium	
Region-based	Medium	
DHT-based	Large	
Voronoi-based	Large	
Ours	Large	Connectivity threshold, NCS dimensionality
TODO: Incomplete list		

TODO: Explain each one

We have built a simple visualisation tool to allow the inspection of computed topologies as the edge costs change in real time, however this is only useful for coordinate spaces that have 3 dimensions or fewer.

5.2.2 Implementation

Browser library and framework

This section details the design of our library based on our requirements, as well as detailing and justifying our design decisions. It is split into and overview and subsections for our two main solutions for these two goals respectively, and a third to discuss server-side design.

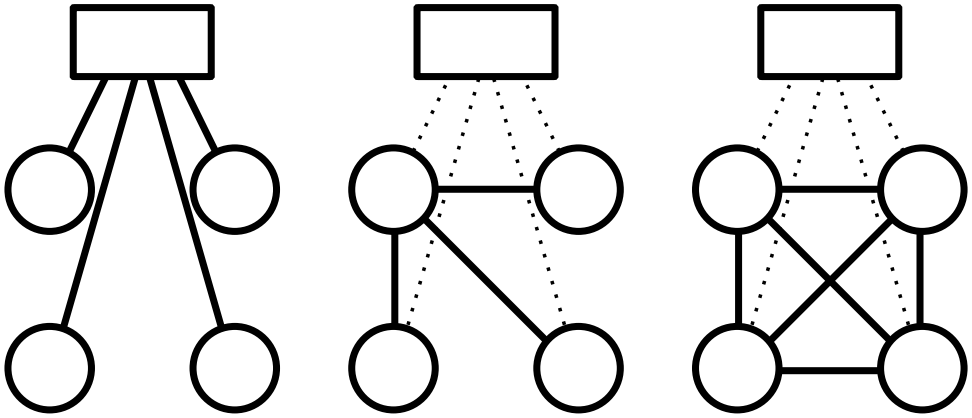


Figure 5.2: Three networking topologies of interest between servers (rectangles) and peers/clients (circles) – client-server model (left), hosted P2P (middle), and full P2P (right)

Overview

At the highest level, our goals are to minimise costs that manifest themselves in server and codebase development and maintenance. As we minimise maintenance by encapsulating general and game-specific functionality in our library, the main requirement in the context of this paper is that our system significantly reduces server costs and these costs scale minimally as we increase the number of players.

The majority of traffic that passes through servers in most games is simply broadcasting information from one client to one or more other clients. We remove the middleman by creating a peer network that allows us to broadcast directly between peers.

What makes our system different from a standard P2P mesh, is that we also connect the peers in a peer network in such a way that peers do not have to maintain too many connections, while at the same time not having to rely on single peers for the fidelity of the communicated data. Currently, this peer network topology is coordinated by the signalling server based on the quality of a connecting between pairwise peers. We discuss this in more detail in this section.

Reducing server costs

The obvious problem with a completely connected P2P approach is that it does not scale for N players as well as a client-server model does, as clients can be resource/bandwidth-weak in comparison to servers that have the resources to maintain N connections. We mitigate this by allowing for alternative peer network topologies (which define how the peers are connected), such as the middle one in figure 5.2 where a peer is designated "host" and acts as a de facto authoritative server, as opposed to a completely connected

topology such as the right one in figure 5.2. This limits the server costs of the game provider to simply acting as a lobby/directory for finding these rooms/groups, but at the same time, the number of players that a host can support is more limited than a standalone server.

A server is still required for signalling purposes to exchange the information required to establish a connection between peers. This low-traffic server connection only needs to be maintained if the peers need to be notified when a new peer joins their network, which translates to joining their game room/area/instance/arena/match/etc. Otherwise it can be severed after a peer network is set up.

Reducing code complexity

Our second goal is maximising ease of development and developer barrier to entry. To do this, we encapsulate all networking functionality behind (*i*) event emitters and (*ii*) shared objects that sync between peers automatically. This is important because we want to avoid code duplication and maintaining strongly related code in more than one place. This is common in client-server applications where changes to client communication requires changes to server communication in parallel, and vice versa. With our system, developers need only maintain peer code.

While figure 5.2 is simplistic, it alludes to a spectrum between a topology with the least number of connections possible — e.g. with a minimum spanning tree (MST) over all peers — and a completely connected topology where each peer is connected to every other peer.

As our peer networks should be able to have arbitrary topologies, it is important that we do not introduce heterogeneity between peers in code as this can quickly become unmanageable. Ultimately, the same peers should be able to switch between an MST topology and a completely connected topology without touching the game logic whatsoever.

Signalling server

Our P2P approach introduces other non-negligible challenges. In a client-server context, there is a stark trade-off between how much game logic the server carries or attests (which increases server costs) and players' ability to cheat by modifying the client. An example of this are games where all physics simulation is done client-side to minimise lag and a player can modify a client to manipulate their position and clip through solids.

In a peer-to-peer context, this problem persists between peers, however the cost of

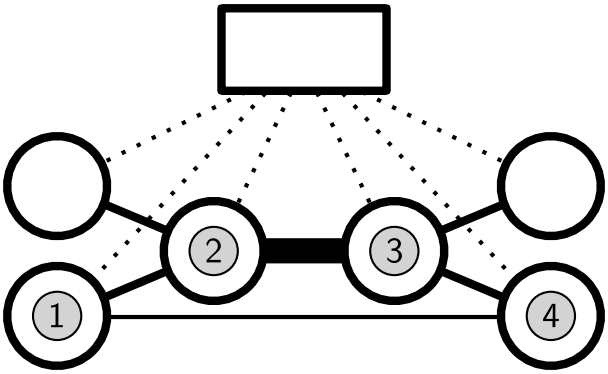


Figure 5.3: An example of a computed MST topology where peers with better connections (② and ③) act as supernodes, and with redundancy (① and ④)

validating game states falls to the clients. As no authoritative servers exist, peers have to decide either manually or autonomously to disconnect and/or blacklist cheating peers by maintaining an array of IPs of past offenders seen first-hand.

On the spectrum between MST and completely-connected, we cannot therefore simply go for MST as it is more efficient. We must introduce redundant connections for two reasons: (*i*) resilience and (*ii*) accountability. Figure 5.3 shows a visual example of this where peer ④ relies on just peer ③ for updates, as peer ③ has a good connection to the left side of the network. We add redundancy by connecting ④ to ① also.

For the first, it is imaginable that a node goes offline for whatever reason, and a new MST must be computed. To avoid the overhead and potential lag in repairing the peer network, redundant connections are an advantage. For the second, if a peer relies on only one other peer to update their global game state, that peer can spoof the game state. While every peer can perform their own validation for impossible game states, or states that imply cheating, there are edge cases where this becomes less obvious when a peer receives realistic but divergent states from two or more other peers.

It is therefore important for the topology to be set up in such a way that each peer can choose to validate data by comparing it between more than one source. A peer that is caught cheating (by e.g. spoofing their in-game position) can then be blacklisted and the network can sever the connection to them. We plan to extend our library to allow this by comparing data across source peers, and flagging peers that diverge from the majority. Additional validation (such as checking if a player's position is changing too fast) is left to the developer and we provide the API for a peer to blacklist them in that case.

Finally, we can take advantage of other information for setting up a peer network topology (the weights for our MST) to reflect that peers with more resources can take on more connections. We do this by periodically checking the latencies between each

pair of peers. From this information, we can optimise peer network topology so that a weak peer does not inadvertently become a supernode. To minimise code complexity, we encapsulate this behaviour in the signalling server.

APIs

Our peer library and signalling server implementations are open source under an MIT license and available on GitHub under *yousefamar/p2p-peer* and *yousefamar/p2p-sig-serv* respectively, and the peer library can be installed through NPM as *p2p-peer* and used as a Node.js module with common browser bundlers such as Browserify and Webpack, or as an ES6 module. Our peer library is 231 SLOC / 7.61 KB of uncompressed, unminified code, and our signalling server is a mere 64 SLOC / 1.53 KB, both pure JavaScript.

P2P communication has begun to see widespread support in modern browsers through WebRTC. While WebRTC was mainly intended to enable audiovisual communication between browsers (such as for video chat applications), it can also be used to communicate arbitrary data. While modern browsers have had this capability for a while, browser-based games have not yet seen widespread adoption of WebRTC for P2P communication between players.

Even outside of games, most applications of WebRTC are limited to chat applications and specialised use cases such as for exchanging metadata in the WebTorrent protocol. WebRTC is not very developer-friendly, and while libraries that simplify P2P communication exist, these are bare-metal and not aimed at game development.

We use WebRTC to enable P2P communication between browsers. Our library exposes a `PeerNetwork` object which handles peer connections and emits events based on different events that happen in the network (such as peers connecting or disconnecting). We allow arbitrary namespacing through *rooms* of which a peer can join multiple.

It also exposes a set of methods that can be called directly if required. These are:

- `signal(event, ...args)` – Sends an event and data to the signalling server
- `join(roomID)` – Joins a room with a particular ID
- `leave(roomID)` – Leaves a room with a particular ID
- `broadcast(event, ...args)` – Sends an event and data to all connected peers
- `async connect(sigServURL)` – Connects to a signalling server

`PeerNetwork` also has three properties; `ownUID` which is the peer's own UID in the network, an array called `peers` which contains instances of `Peer` for all connected peers, and an array called `rooms` which will be explained shortly.

One can `.disconnect()` from or `.send(event, data)` directly to each `Peer`, and on the other side, peers will emit events based on what is sent that can be listened to. These events do not need to be defined anywhere beforehand.

As an added layer of abstraction over network events, each `room` contains an `eventEmitter` `syncedData` object which implements an Observer design pattern such that any changes made to this object (or any nested objects at any depth) emit an event distinct to a property's path in this object. The value, along with the path of the added/modified property's path (e.g. `.foo.bar`) is propagated through the peer network and all peers in a room can expect to see the property at that path updated. For added efficiency, we only broadcast data that has been modified. We also keep track of which data has been set by which peer for data ownership and to prevent broadcast storms in the peer network.

Namespacing within the object and how rooms are organised is left to the developer to define. As mentioned before, rooms can map directly to an arena instance in a MOBA context for example. Another possibility is rooms mapping to a 2D or 3D spatial grid where players join and leave rooms seamlessly as they navigate through the game environment, prioritising connections to the players who are closest to them in the game.

Similarly, rooms can be "nested" into a tree structure for optimisation purposes, such as for limiting update frequency for peers that are further away from each other in-game (e.g. position updates) creating network Levels of Detail (LODs). This is especially powerful in an MMO context.

While the organisation of rooms is left to developers, and the current abstractions are not just game-centric, we plan to provide example code for the most common use cases such as these. We also plan to survey the most common kinds of data transmitted in networked browser games (e.g. position updates vs events like jumping) and build functionality for doing this seamlessly.

TODO: Write up implementation of a peer network topology targeted at large-scale multiplayer games

TODO: Write up implementation of other common P2P models

TODO (maybe): Write up implementation of anti-cheat measures

5.2.3 Evaluation

This section describes a simple evaluation of our method and library. Our aim is to demonstrate how we achieve significant reductions in server traffic, thereby lowering required bandwidth and sever costs.

Topologies

Sensitivity analysis on link quality (mean/variance latency) for each

Pairwise latency distribution for each

Scalability

Setup and Method

We deploy up to eight client/peers and two servers. The first server acts as a signalling server for the P2P implementation, and the second exposes a WebSocket endpoint that allows clients to broadcast messages to other clients in the same room through the server.

The clients/peers then make use of our library to set a random floating point number on a set room's `syncedData` object every 100 milliseconds. Our library propagates these values through the network.

We use `nethogs` to then measure the traffic in KB/s for the server processes. We collect traffic data for several minutes before incrementing the number of clients/peers and repeating the process. The measurements for the P2P implementation and the client-server implementation are done separately.

Results

Figure 5.4 shows how the mean server network I/O changes as we increase the number of clients/peers. While the client-server implementation scales exponentially as we add more clients, the server traffic for our P2P implementation increases linearly and very slowly.

This is because the P2P signalling server only sends periodic heartbeats to each peer to check if a connection is still alive. The data does not need to be sent through the client and we shift the burden of reaching other peers to the peer. For few peers (<30), modern devices can cope well with a completely connected network. Beyond that, our optimised topologies make a significant difference, which we aim to demonstrate in future work.

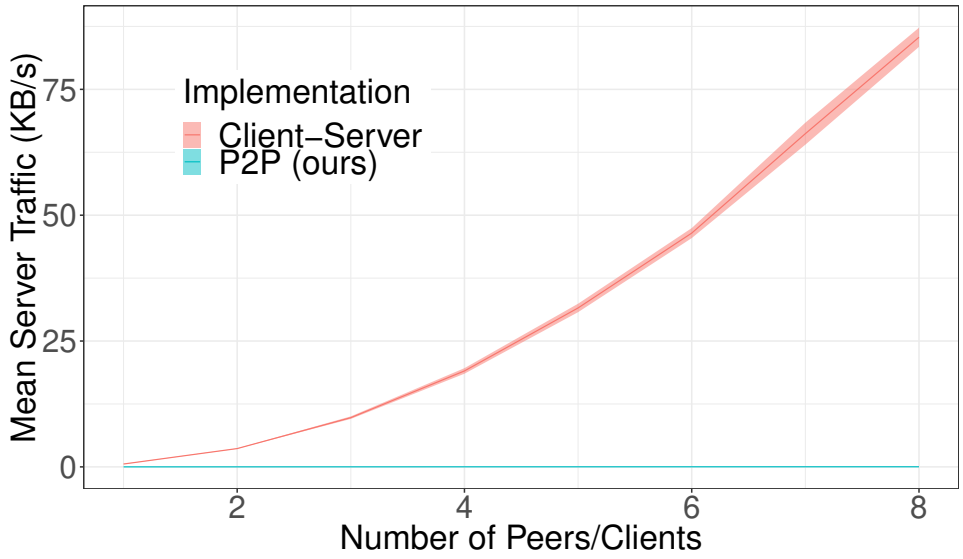


Figure 5.4: Mean server network traffic (0.95 confidence interval) against number of clients/peers for a traditional client-server model versus our optimised P2P version

TODO: Traffic against network size for each

Validation

TODO (maybe): Evaluation of anti-cheat measures

5.2.4 Discussion and summary

We have identified server costs and development overhead as two deficiencies in multiplayer browser game development that create a barrier to entry for independent game developers. We have addressed these deficiencies by introducing a library that takes advantage of modern HTML5 APIs to simplify P2P communication for games and make it scalable. Finally, we have empirically demonstrated the advantages that our library provides by evaluating it against the standard approach.

While our library can be sufficient for low-throughput networked browser games, there are some cases where games might need unreliable, but higher-throughput, streaming communication (cf. UDP). For this, our event-based system may be unsuitable. A natural extension of our library would be more bare-metal APIs for that type of data.

Currently, the signalling server orchestrates the topology of the peer network, introducing some centralisation. We can decentralise our system even further by instead employing a more peer-centric method, where e.g. peers dynamically reconnect to better

peers over time, thus eventually converging on optimal networks. This minimises points of failure.

Finally, we aim to extend the scalability of our system such that it would be suitable for use in an MMO where a single peer may need to communicate with a very large number of peers can cannot be expected to broadcast data to all of them. To do this requires further investigation and evaluation of peer network topologies, however can yield significant cost benefits.

Future work, general-purpose, secure serverless platform in the browser

Chapter 6

Conclusion

TODO: Rewrite and expand

We first presented a system for efficiently augmenting token-based access control with privacy-awareness without significantly impacting performance or utility. We described our implementation and demonstrated its practicality through experimental evaluations in terms or privacy gains and performance on cheap hardware.

Then we presented the design and implementation of our approach that exploits cached partition/queue lengths to predict the service rates and the expected message response times. Combining this information with an effective heuristic to pick the best-of-two random partitions enables our system to outperform other common partitioning approaches to reduce response times by 31.08% with virtually no extra overhead.

This research shows that it is possible to augment access control systems with privacy-awareness at little cost to utility and performance. We intend to show that the same can be said for job distribution systems.

In future, it may be possible to use these techniques to build nested privacy contexts that can educate users on privacy "gradients" between nodes and automate the flow of data based on user preferences.

Bibliography

[1] *4.8.2 The iframe element – HTML5*. URL: https://www.w3.org/TR/2011/WD-html5-20110525/the-iframe-element.html#attr-iframe-sandbox (visited on 03/11/2016).

[2] Mohammad Alizadeh et al. "CONGA: Distributed congestion-aware load balancing for datacenters". In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. ACM. 2014, pp. 503–514.

[3] Hazim Almuhimedi et al. "Your location has been shared 5,398 times!: A field study on mobile app privacy nudging". In: *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM. 2015, pp. 787–796.

[4] Yousef Amar. *Trace Analysis Scripts*. https://github.com/yousefamar/trace-analysis-scripts. [Online; accessed 24-February-2018]. 2018.

[5] Yousef Amar et al. "An Analysis of Home IoT Network Traffic and Behaviour". In: *arXiv preprint arXiv:1803.05368* (2018).

[6] Derek Mc Auley, Richard Mortier, and James Goulding. "The dataware manifesto". In: *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*. IEEE. 2011, pp. 1–6.

[7] Steve Benford and Lennart Fahlén. "A spatial model of interaction in large virtual environments". In: *Proceedings of the Third European Conference on Computer-Supported Cooperative Work 13–17 September 1993, Milan, Italy ECSCW'93*. Springer. 1993, pp. 109–124.

[8] Michele Bezzi. "An information theoretic approach for privacy metrics." In: *Trans. Data Privacy* 3.3 (2010), pp. 199–215.

[9] Michele Bezzi. "Expressing privacy metrics as one-symbol information". In: *Proceedings of the 2010 EDBT/ICDT Workshops*. ACM. 2010, p. 29.

[10] Upendra Bhoi, Purvi N Ramanuj, et al. "Enhanced max-min task scheduling algorithm in cloud computing". In: *International Journal of Application or Innovation in Engineering and Management (IJAIEM)* 2.4 (2013), pp. 259–264.

[11] Zack Bloom. *Serverless Performance: Cloudflare Workers, Lambda and Lambda@Edge*. 2018. URL: https://blog.cloudflare.com/serverless-performance-comparison-workers-lambda/ (visited on 07/21/2019).

[12] *BRISSKit - Biomedial Research Software as a Service*. 2015. URL: https://www.brisskit.le.ac.uk/ (visited on 03/09/2016).

[13] Anthony Brown, Richard Mortier, and Tom Rodden. "MultiNet: reducing interaction overhead in domestic wireless networks". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2013, pp. 1569–1578.

[14] Loc Bui, R Srikant, and Alexander Stolyar. "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing". In: *INFOCOM 2009, IEEE*. IEEE. 2009, pp. 2936–2940.

[15] Eliya Buyukkaya and Maha Abdallah. "Efficient triangulation for p2p networked virtual environments". In: *Multimedia Tools and Applications* 45.1-3 (2009), pp. 291–312.

[16] Eliya Buyukkaya, Maha Abdallah, and Romain Cavagna. "VoroGame: a hybrid P2P architecture for massively multiplayer games". In: *2009 6th IEEE Consumer Communications and Networking Conference*. Ieee. 2009, pp. 1–5.

[17] Supriyo Chakraborty et al. "ipShield: a framework for enforcing context-aware privacy". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 2014, pp. 143–156.

[18] Amir Chaudhry et al. "Personal Data: Thinking Inside the Box". In: *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives*. AA '15. Aarhus, Denmark: Aarhus University Press, 2015, pp. 29–32. DOI: 10.7146/aahcc.v1i1.21312. URL: http://dx.doi.org/10.7146/aahcc.v1i1.21312.

[19] Terence Chen et al. "How much is too much? Leveraging ads audience estimation to evaluate public profile uniqueness". In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2013, pp. 225–244.

[20] Yang Chen et al. "Pharos: accurate and decentralised network coordinate system". In: *IET communications* 3.4 (2009), pp. 539–548.

[21] Yang Chen et al. "Phoenix: Towards an accurate, practical and decentralized network coordinate system". In: *International Conference on Research in Networking*. Springer. 2009, pp. 313–325.

[22] *Content Security Policy Level 2*. 2015. URL: https://www.w3.org/TR/CSP2/ (visited on 03/11/2016).

[23] Adam Conway. *Android getting "DNS over TLS" support to stop ISPs from knowing what websites you visit*. https://www.xda-developers.com/android-dns-over-tls-website-privacy/. [Online; accessed 24-February-2018]. 2017.

[24] *D3.js - Data-Driven Documents*. URL: https://d3js.org/ (visited on 03/11/2016).

[25] Frank Dabek et al. "Vivaldi: A decentralized network coordinate system". In: *ACM SIGCOMM Computer Communication Review*. Vol. 34. 4. ACM. 2004, pp. 15–26.

[26] Tore Dalenius. "Finding a needle in a haystack or identifying anonymous census records". In: *Journal of official statistics* 2.3 (1986), p. 329.

[27] Raluca Diaconu and Joaquín Keller. "Kiwano: A scalable distributed infrastructure for virtual worlds". In: *2013 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2013, pp. 664–667.

[28] Claudia Diaz, Carmela Troncoso, and Andrei Serjantov. "On the impact of social network profiling on anonymity". In: *Privacy Enhancing Technologies*. Springer. 2008, pp. 44–62.

[29] *Docker*. 2016. URL: `https://www.docker.com/` (visited on 03/11/2016).

[30] *Docker Registry v2 Bearer token specification*. URL: `https://docs.docker.com/registry/spec/auth/jwt/` (visited on 03/11/2016).

[31] Cynthia Dwork. "Differential privacy". In: *Automata, languages and programming*. Springer, 2006, pp. 1–12.

[32] Daniel E Eisenbud et al. "Maglev: A Fast and Reliable Software Network Load Balancer." In:

[33] *Electron*. URL: `http://electron.atom.io/` (visited on 03/11/2016).

[34] Blake Embrey. *Express-style path to regexp*. `https://github.com/pillarjs/path-to-regexp`. 2016. (Visited on 06/29/2016).

[35] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. "Security analysis of emerging smart home applications". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 636–654.

[36] Davide Frey et al. "Solipsis: A decentralized architecture for virtual environments". In: *1st International Workshop on Massively Multiuser Virtual Environments*. 2008.

[37] Benjamin Fung et al. "Privacy-preserving data publishing: A survey of recent developments". In: *ACM Computing Surveys (CSUR)* 42.4 (2010), p. 14.

[38] Game Developers Conference (GDC). *State of the Game Industry 2017*. Tech. rep. 2017.

[39] Game Developers Conference (GDC). *State of the Game Industry 2018*. Tech. rep. 2018.

[40] John S Gilmore and Herman A Engelbrecht. "A survey of state persistency in peer-to-peer massively multiplayer online games". In: *IEEE Transactions on Parallel and Distributed Systems* 23.5 (2011), pp. 818–834.

[41] Alex Glikson, Stefan Nastic, and Schahram Dustdar. "Deviceless edge computing: extending serverless computing to the edge of the network". In: *Proceedings of the 10th ACM International Systems and Storage Conference*. ACM. 2017, p. 28.

[42] Mark S Gordon et al. "COMET: Code Offload by Migrating Execution Transparently." In: *OSDI*. Vol. 12. 2012, pp. 93–106.

[43] Chris Greenhalgh and Steve Benford. "MASSIVE: a distributed virtual reality system incorporating spatial trading". In: *Proceedings of 15th International Conference on Distributed Computing Systems*. IEEE. 1995, pp. 27–34.

[44] Ulrich Greveler et al. "Multimedia content identification through smart meter power usage profiles". In: *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2012, p. 1.

[45] Parul Gupta and Tara Javidi. "Towards throughput and delay optimal routing for wireless ad-hoc networks". In: *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*. IEEE. 2007, pp. 249–254.

[46] Hamed Haddadi et al. "Personal Data: Thinking Inside the Box". In: *Proc. 5th Decennial ACM Aarhus Conference: Critical Alternatives*. Aarhus, Denmark, Aug. 2015.

[47] Jun Han et al. "Accomplice: Location inference using accelerometers on smartphones". In: *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*. IEEE. 2012, pp. 1–9.

[48] Tom Hardin. *2018 Digital Trend: Microservices*. 2018. URL: `https://blog.g2crowd.com/blog/trends/digital-platforms/2018-dp/microservices/` (visited on 05/14/2019).

[49] Danny Harnik and Eliad Tsfadia. *Impressions of Intel®SGX performance – Danny Harnik – Medium*. 2017. URL: `%5Curl%7Bhttps://medium.com/@danny_harnik/impressions-of-intel-sgx-performance-22442093595a%7D` (visited on 10/13/2018).

[50] Keqiang He et al. "Presto: Edge-based load balancing for fast datacenter networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 465–478.

[51] Scott Hendrickson et al. "Serverless computation with openlambda". In: *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. 2016.

[52] Christian E Hopps. "Analysis of an equal-cost multi-path algorithm". In: (2000).

[53] Arie Hordijk and Ger Koole. "On the optimality of the generalized shortest queue policy". In: *Probability in the Engineering and Informational Sciences* 4.4 (1990), pp. 477–487.

[54] David J Houck. "Comparison of policies for routing customers to parallel queueing systems". In: *Operations Research* 35.2 (1987), pp. 306–310.

[55] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. "VON: a scalable peer-to-peer network for virtual environments". In: *IEEE Network* 20.4 (2006), pp. 22–31.

[56] Shun-Yun Hu et al. "A spatial publish subscribe overlay for massively multiuser virtual environments". In: *2010 International Conference on Electronics and Information Engineering*. Vol. 2. IEEE. 2010, pp. V2–314.

[57] Identity Theft Resource Center. *ITRC Breach Statistics 2005 - 2015*. 2016. URL: `http://www.idtheftcenter.org/images/breach/2005to2015multiyear.pdf` (visited on 03/09/2016).

[58] Laura Itzel et al. "The quest for meaningful mobility in massively multi-user virtual environments". In: *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*. IEEE Press. 2011, p. 15.

[59] Josh James. *Data Never Sleeps 2.0*. 2014. URL: `https://www.domo.com/blog/2014/04/data-never-sleeps-2-0/` (visited on 03/09/2016).

[60] Pravin K Johri. “Optimality of the shortest line discipline with state-dependent service rates”. In: *European Journal of Operational Research* 41.2 (1989), pp. 157–161.

[61] State of JS. *The State of JavaScript 2018: Front-end Frameworks - Overview*. 2019. URL: `https://2018.stateofjs.com/front-end-frameworks/overview/` (visited on 07/20/2019).

[62] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[63] Georgios Kalogridis et al. “Privacy for smart meters: Towards undetectable appliance load signatures”. In: *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE. 2010, pp. 232–237.

[64] Kleomenis Katevas, Hamed Haddadi, and Laurissa Tokarchuk. “Poster: Sensingkit: A multi-platform mobile sensing framework for large-scale experiments”. In: *Proceedings of the 20th annual international conference on Mobile computing and networking*. ACM. 2014, pp. 375–378.

[65] Naga Katta et al. “Hula: Scalable load balancing using programmable data planes”. In: *Proceedings of the Symposium on SDN Research*. ACM. 2016, p. 10.

[66] Mayanka Katyal and Atul Mishra. “A comparative study of load balancing algorithms in cloud computing environment”. In: *arXiv preprint arXiv:1403.6918* (2014).

[67] Mayanka Katyal and Atul Mishra. “A comparative study of load balancing algorithms in cloud computing environment”. In: 2014.

[68] Joaquín Keller and Gwendal Simon. “Toward a peer-to-peer shared virtual reality”. In: *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*. IEEE. 2002, pp. 695–700.

[69] Roelof Kemp et al. “Cuckoo: a computation offloading framework for smartphones”. In: *International Conference on Mobile Computing, Applications, and Services*. Springer. 2010, pp. 59–79.

[70] Younghun Kim, Edith C-H Ngai, and Mani B Srivastava. “Cooperative state estimation for preserving privacy of user behaviors in smart grid”. In: *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*. IEEE. 2011, pp. 178–183.

[71] Bjorn Knutsson et al. “Peer-to-peer support for massively multiplayer games”. In: *IEEE INFOCOM 2004*. Vol. 1. IEEE. 2004.

[72] T Kokilavani, DI George Amalarethinam, et al. “Load balanced min-min algorithm for static meta-task scheduling in grid computing”. In: *International Journal of Computer Applications* 20.2 (2011), pp. 43–49.

[73] J Zico Kolter and Matthew J Johnson. “REDD: A public data set for energy disaggregation research”. In: 2011.

[74] KR Krishnan. “Joining the right queue: a state-dependent decision rule”. In: *IEEE Transactions on Automatic Control* 35.1 (1990), pp. 104–108.

[75] Santosh Kulkarni. "Badumna network suite: A decentralized network engine for massively multiplayer online applications". In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE. 2009, pp. 178–183.

[76] Eyal de Lara et al. "Hierarchical Serverless Computing for the Mobile Edge". In: *IEEE/ACM Symposium on Edge Computing*. 2016.

[77] Dhinesh Babu LD and P Venkata Krishna. "Honey bee behavior inspired load balancing of tasks in cloud computing environments". In: *Applied Soft Computing* 13.5 (2013), pp. 2292–2303.

[78] Sergey Legtchenko, Sébastien Monnet, and Ga¨el Thomas. "Blue Banana: resilience to avatar mobility in distributed MMOGs". In: *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE. 2010, pp. 171–180.

[79] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-closeness: Privacy beyond k-anonymity and l-diversity". In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE. 2007, pp. 106–115.

[80] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[81] Zhen Liu and Rhonda Righter. "Optimal load balancing on distributed homogeneous unreliable processors". In: *Operations Research* 46.4 (1998), pp. 563–573.

[82] Zhen Liu and Don Towsley. "Optimality of the round-robin routing policy". In: *Journal of applied probability* 31.2 (1994), pp. 466–475.

[83] Ashwin Machanavajjhala et al. "l-diversity: Privacy beyond k-anonymity". In: *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE. 2006, pp. 24–24.

[84] Yuyi Mao et al. "A survey on mobile edge computing: The communication perspective". In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2322–2358.

[85] Peyton Maynard-Koran. *Fixing the Internet for Real Time Applications: Part I*. 2015. URL: `https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-i` (visited on 07/18/2019).

[86] Peyton Maynard-Koran. *Fixing the Internet for Real Time Applications: Part II*. 2016. URL: `https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-ii` (visited on 07/18/2019).

[87] Peyton Maynard-Koran. *Fixing the Internet for Real Time Applications: Part III*. 2016. URL: `https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-iii` (visited on 07/18/2019).

[88] Frank McSherry and Ratul Mahajan. "Differentially-private network trace analysis". In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 123–134.

[89] Ronald Menich and Richard F Serfozo. "Optimality of routing and servicing in dependent parallel processing systems". In: *Queueing Systems* 9.4 (1991), pp. 403–418.

[90] MongoDB. *The MongoDB Database*. https://www.mongodb.com/. Apr. 2017.

[91] Yves-Alexandre de Montjoye et al. "openpds: Protecting the privacy of metadata through safeanswers". In: *PloS one* 9.7 (2014), e98790.

[92] Katherine L Morse et al. *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine, 1996.

[93] *Mydex*. 2012. URL: https://data.gov.uk/library/mydex (visited on 03/09/2016).

[94] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. "Can homomorphic encryption be practical?" In: *Proc. ACM Cloud Computing Security Workshop*. 2011, pp. 113–124.

[95] Mohammad Naghshvar and Tara Javidi. "Opportunistic routing with congestion diversity in wireless multi-hop networks". In: *INFOCOM, 2010 Proceedings IEEE*. IEEE. 2010, pp. 1–5.

[96] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE. 2008, pp. 111–125.

[97] Michael J Neely and Rahul Urgaonkar. "Optimal backpressure routing for wireless networks with multi-receiver diversity". In: *Ad Hoc Networks* 7.5 (2009), pp. 862–881.

[98] Mark Nottingham. *Internet protocols are changing*. https://blog.apnic.net/2017/12/12/internet-protocols-changing/. [Online; accessed 24-February-2018]. 2017.

[99] *NW.js*. URL: http://nwjs.io/ (visited on 03/11/2016).

[100] Eliza Papadopoulou et al. "Enabling data subjects to remain data owners". In: *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 2015, pp. 239–248.

[101] BSI PAS. "212:2016 Automatic resource discovery for the Internet of Things - Specification". In: *British Standards Institution* (2016).

[102] Kenny Paterson. *Industry Concerns about TLS 1.3*. https://www.ietf.org/mail-archive/web/tls/current/msg21278.html. [Online; accessed 24-February-2018]. 2016.

[103] Charith Perera et al. "Context aware computing for the internet of things: A survey". In: *IEEE communications surveys & tutorials* 16.1 (), pp. 414–454.

[104] The Bro Project. *The Bro Network Security Monitor*. https://www.bro.org/. [Online; accessed 24-February-2018]. 2014.

[105] Theo de Raadt. *pledge - a new mitigation mechanism*. http://www.openbsd.org/papers/hackfest2015-pledge/. 2015. (Visited on 04/19/2016).

[106] Andrea W Richa, M Mitzenmacher, and R Sitaraman. "The power of two random choices: A survey of techniques and results". In: *Combinatorial Optimization* 9 (2001), pp. 255–304.

[107] Eleanor Rieffel et al. “Secured histories: computing group statistics on encrypted data while preserving individual privacy”. In: *arXiv preprint arXiv:1012.2152* (2010).

[108] Zvi Rosberg and Parviz Kermani. “Customer routing to different servers with complete information”. In: *Advances in Applied Probability* 21.4 (1989), pp. 861–882.

[109] Arne Schmieg et al. “pSense-Maintaining a dynamic localized peer-to-peer structure for position based multicast in games”. In: *2008 Eighth International Conference on Peer-to-Peer Computing*. IEEE. 2008, pp. 247–256.

[110] Andrea Sciarrone et al. “Context awareness over transient clouds”. In: *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE. 2015, pp. 1–5.

[111] Rayman Preet Singh et al. “TussleOS: Managing Privacy Versus Functionality Trade-Offs on IoT Devices”. In: *ACM SIGCOMM Computer Communication Review* 46.3 (2016).

[112] StatCounter Global Stats. *Browser market share*. 2019. URL: `https://netmarketshare.com/browser-market-share.aspx` (visited on 07/20/2019).

[113] StatCounter Global Stats. *Browser Market Share Worldwide*. 2019. URL: `http://gs.statcounter.com/browser-market-share#monthly-201811-201811-bar` (visited on 07/20/2019).

[114] StatCounter Global Stats. *Desktop vs Mobile vs Tablet Market Share Worldwide*. 2019. URL: `http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet` (visited on 07/20/2019).

[115] Latanya Sweeney. “k-anonymity: A model for protecting privacy”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[116] Latanya Sweeney. “Simple demographics often identify people uniquely”. In: *Health (San Francisco)* 671 (2000), pp. 1–34.

[117] *Unikernels - Rethinking Cloud Infrastructure*. 2016. URL: `http://unikernel.org/` (visited on 03/11/2016).

[118] Mathy Vanhoef and Frank Piessens. “Key reinstallation attacks: Forcing nonce reuse in WPA2”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1313–1328.

[119] Mathy Vanhoef et al. “Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms”. In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, pp. 413–424.

[120] W3Techs. *Usage of server-side programming languages for websites*. 2019. URL: `https://w3techs.com/technologies/overview/programming_language/all` (visited on 06/19/2019).

[121] Isabel Wagner and David Eckhoff. “Technical privacy metrics: a systematic survey”. In: *arXiv preprint arXiv:1512.00327* (2015).

[122] Frank Wang et al. “Sieve: cryptographically enforced access control for user data in untrusted clouds”. In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016, pp. 611–626.

[123] Wayne Winston. “Optimality of the shortest line discipline”. In: *Journal of Applied Probability* 14.1 (1977), pp. 181–189.

[124] Amir Yahyavi and Bettina Kemme. “Peer-to-peer architectures for massively multiplayer online games: A survey”. In: *ACM Computing Surveys (CSUR)* 46.1 (2013), p. 9.

[125] Lei Ying et al. “On combining shortest-path and back-pressure routing over multihop wireless networks”. In: *IEEE/ACM Transactions on Networking (TON)* 19.3 (2011), pp. 841–854.

Glossary

AOI Area-of-Interest. 89

AR Augmented Reality. 98

AWS Amazon Web Services. 84

CDN Content Delivery Network. 85

CRUD Create, Read, Update, and Delete. 86

CSP Content Security Policy. 86

CSR Client-Side Rendering. 10, 86, 87

data processor As per GDPR, "an entity which processes personal data on behalf of the controller". 1, 2, 16

data source A sensor, device, service, or database that produces data. 1

data subject An entity whose data is processed. 2

DHT Distributed Hash Table. 10, 89

edge computing Processing done near the edge of a network (e.g. on smart devices and edge devices such as IoT devices) as opposed to in the cloud. 3

EMST Euclidean Minimum Spanning Tree. 96

FPS First-Person Shooter. 98

IM Interest Management. 89, 91

IoT Internet of Things. 1, 3, 7, 11, 16, 44, 84, 87

job An instantiation of arbitrary computation that requires a certain amount of work for execution to be complete. 16

job completion time The time it takes for a job to complete from arrival into the system to result response. 17

MMOG Massively Multiplayer Online Game. 87, 89, 97, 98

MOBA Multiplayer Online Battle Arena. 98

MST Minimum Spanning Tree. 96

NCS Network Coordinate Systems. 91, 95

P2P Peer-to-Peer. 10, 89

privacy-aware Systems that are privacy-aware can operate in a privacy-preserving manner. 2

PWA Progressive Web App. 10, 86

REST Representational State Transfer. 44, 86, 87

rts Real-Time Strategy. 98

service provider Companies the provide an online service such as social networking, web search, video streaming, etc. 2

SPA Single-Page Application. 10, 86

VE Virtual Environment. 89, 91, 94, 97, 98

VPS Virtual Private Server. 84

VR Virtual Reality. 87

work Computation undertaken to execute a job in part or in its entirety on a particular compute node. 16

XSS Cross-Site Scripting. 86