Profiling Large-scale Live Video Streaming and Distributed Applications

by

Jie Deng

Ph.D.

School of Electronic Engineering and Computer Science
Queen Mary University of London
United Kingdom

June 2018

TO MY FAMILY

Abstract

Today, distributed applications run at data centre and Internet scales, from intensive data analysis, such as MapReduce; to the dynamic demands of a worldwide audience, such as YouTube. The network is essential to these applications at both scales. To provide adequate support, we must understand the full requirements of the applications, which are revealed by the workloads. In this thesis, we study distributed system applications at different scales to enrich this understanding.

Large-scale Internet applications have been studied for years, such as social networking service (SNS), video on demand (VoD), and content delivery networks (CDN). An emerging type of video broadcasting on the Internet featuring crowdsourced live video streaming has garnered attention allowing platforms such as Twitch to attract over 1 million concurrent users globally. To better understand Twitch, we collected real-time popularity data combined with metadata about the contents and found the broadcasters rather than the content drives its popularity. Unlike YouTube and Netflix where content can be cached, video streaming on Twitch is generated instantly and needs to be delivered to users immediately to enable real-time interaction. Thus, we performed a large-scale measurement of Twitchs content location revealing the global footprint of its infrastructure as well as discovering the dynamic stream hosting and client redirection strategies that helped Twitch serve millions of users at scale.

We next consider applications that run inside the data centre. Distributed computing applications heavily rely on the network due to data transmission needs and the scheduling of resources and tasks. One successful application, called Hadoop, has been widely deployed for Big Data processing. However, little work has been devoted to understanding its network. We found the Hadoop behaviour is limited by hardware resources and processing jobs presented. Thus, after characterising the Hadoop traffic on our testbed

with a set of benchmark jobs, we built a simulator to reproduce Hadoops job traffic. With the simulator, users can investigate the connections between Hadoop traffic and network performance without additional hardware cost. Different network components can be added to investigate the performance, such as network topologies, queue policies, and transport layer protocols.

In this thesis, we extended the knowledge of networking by investigated two widely-used applications in the data centre and at Internet scale. We (*i*)studied the most popular live video streaming platform Twitch as a new type of Internet-scale distributed application revealing that broadcaster factors drive the popularity of such platform, and we (*ii*)discovered the footprint of Twitch streaming infrastructure and the dynamic stream hosting and client redirection strategies to provide an in-depth example of video streaming delivery occurring at the Internet scale, also we (*iii*)investigated the traffic generated by a distributed application by characterising the traffic of Hadoop under various parameters, (*iv*)with such knowledge, we built a simulation tool so users can efficiently investigate the performance of different network components under distributed application

Acknowledgments

I would like to extend thanks to the many people, who so generously contributed to the work presented in this thesis.

I would like to thank my supervisor, Prof. Steve Uhlig, for his rigorous attitude on research. Along with patients and encouragements, he has provided me with a perfect example on the research journey. I would also like to thank Dr. Felix Cuadrado and Dr. Gareth Tyson. I have been fortunate to have them, not just as my supervisors, but more as a team and friends. I would also like to thank all the members of the Networks groups who helped me in my study. In particular, I would like to thank Kishan Patel, Timm Böttger for helping me manage the hardware which is very important to my research, Dr. Richard Clegg and Dr. Ignacio Castro for their precious time and suggestions on my research work.

I must express my gratitude to Dr. Yongxu Zhu, my wife, for her continued support and encouragement. By studying with her but in different areas, she opened my eye and helped me find the universal principle in research: don't bind to the topic but looking for the true essence. Special thanks to my family including my father, my mother, my father-in-law, my mother-in-law and my newborn son. They are the light and power of my life when I am struggling with the darkness and pain.

Completing this work would have been impossible without the support and friendship provided by the other members from the School of Electronic Engineering and Computer Science, especially the systems support staff. I am indebted to them for their help.

Finally, I would like to thank Queen Mary University of London, not only for providing the funding which allowed me to undertake this research, but also for giving me the opportunity to attend conferences and meet so many interesting people.

Publications

During the compilation of this thesis, the following related articles have been published by the author.

Based on work in Chapter 4:

- Deng, J., Cuadrado, F., Tyson, G. and Uhlig, S.,2015. Behind the Game: Exploring the Twitch Streaming Platform. In Network and Systems Support for Games (NetGames), 2015 14th Annual Workshop on. IEEE.

Based on work in Chapter 5:

- Deng, J., Tyson, G., Cuadrado, F. and Uhlig, S., 2017, March. Internet scale user-generated live video streaming: The Twitch case. In International Conference on Passive and Active Network Measurement (pp. 60-71). Springer, Cham.

Based on work in Chapter 6 and Chapter 7:

- Deng, J., Tyson, G., Cuadrado, F. and Uhlig, S., 2017, June. Keddah: Capturing Hadoop Network Behaviour. In International Conference on Distributed Computing Systems (ICDCS), 2017 37th IEEE International Conference on. IEEE.

- (Under review) Deng, J., Tyson, G., Cuadrado, F. and Uhlig, S., 2017. Keddah: Network Evaluation Powered by Simulating Distributed Application Traffic. ACM Transactions on Modeling and Computer Simulation (TOMACS).

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AQM	Active Queue Management
AS	Autonomous System
CDN	Content Delivery Network
DCN	Data Center Network
DNS	domain name service
ECMP	Equal Cost Multi Path
IPTV	Internet Protocol Television
NFV	Network Function Virtualization
P2P	Peer to Peer
PoP	Point of Presence
QCN	Quantized Congestion Notification
QoS	Quality of Service
RTT	Round Trip Time
SDN	Software Defined Networking
SNS	Social Networking Service
TCP	Transmission Control Protocol
UGC	User Generated Content
VOD	Video On Demand
VoIP	Voice over Internet Protocol

Part I

Introduction

Chapter 1

Introduction

Networks are essential for todays global demands in the exchange of information. From local access networks, where users are located, to data centres, where content is stored, the network is everywhere supporting the delivery of information. However, with an increasing number of applications, networks must be more flexible and smarter to handle many different kinds of traffic. Especially when used in distributed application environments, the network is used intensively. Distributed applications run on multiple computers, execute tasks on multiple systems simultaneously, and use the network to coordinate and communicate.

1.1 Motivation

Distributed applications need to communicate with multiple servers or devices on the same network from any geographical location. The distributed nature of the applications refer to data or computing tasks being spread out over multiple computers. Though various techniques have been proposed to simplify the process of spreading resources, the network remains essential for distributed applications to coordinate tasks and transport data over multiple machines. For example, virtualisation makes the shifting of computing

resources more manageable, but also needs the network to cope dynamically with the data flows. To address this need for flexibility in the network, the networking community proposed the introduction of many new features including enhanced reliability, security, and robustness. For example, software-defined networking (SDN) and network function virtualisation (NFV) have been recently proposed so that networking functions can be operated dynamically to cope with different applications.

However, with so many different technologies available, the network still needs to be configured according to the expected traffic, and proper configuration of the network is necessary to help with the performance of traffic delivery. Taking data centre networking (DCN) as an example,Singla *et al.* [1] pointed out that traffic patterns are still a key factor in measuring DCN performance and guiding the traffic engineering mechanisms, queue policies, and congestion control protocols. However, applications have different traffic patterns and performance metrics, for example, (*i*) video content has high requirements for throughput and delay, but not packet loss, (*ii*) instant messaging has high requirements on packet loss and latency, (*iii*) web requests [2] require low delay in network paths, (*iv*) grid and high-performance computing require high reliability and fault tolerance [3] from the network, and (*v*) virtualization environments require networks to be more dynamic in response to varying applications [4]. Thus, configuring the network properly is challenging as it calls for an in-depth understanding of application behaviours, requirements, and traffic bottlenecks.

Profiling application traffic and understanding application behaviour is essential for determining the appropriate network set-up and configuration. Profiling applications is a complicated process due to the diversity of application behaviours, and applications have differing popularities. People tend to study popular applications that provide significant impacts on the network, such as the work done characterising the traffic workload for both applications in the data centre [2, 4–8] where the computational, storage and network resources are interconnected, and applications on the Internet [9–13] where end users are more involved. Many observers discovered that different traffic patterns are generated

from applications. Ersoz *et al.* [6] observed the traffic of applications based on web requests and server replies and characterised the distribution of different workloads. Roy *et al.* [14] exposed the traffic patterns of applications including the web, caching, and Hadoop inside a data centre hosting a social network platform, and found various traffic arrival patterns and scales for different applications. Similarly, on the Internet, Rao *et al.* [15] found that Netflix and YouTube adapt streaming strategies depending on the application and container (Silverlight, Flash, or HTML5) used. With a change of application or container, the network may be significantly impacted.

Thus, there are no shortcuts to finding application traffic patterns as applications must be analysed individually. Because the measurement methodology matters for traffic characteristics, especially on flow size and concurrent flows [16], data-driven studies are the most reliable approach for exploring application behaviours. New applications need to be analysed with a reliable method to reveal their network behaviour. In this thesis, we extend the understanding of distributed applications so that new insights can be found while choosing networking technologies accordingly.

1.2 Problem Statement

The behaviour of many important applications which recently emerged has not yet been investigated, such as the distributed application and user-generated live video streaming. Without a thorough understanding of application behaviour, the network cannot deliver the application traffic better than a best-effort. This thesis addresses this deficiency and commits to extending the understanding of such applications. We identify popular and important applications at different scales and bridge the gap from understanding the application behaviour to improving application traffic delivery.

The profiled applications from previous studies [2, 4–13] fall into the two categories of small-scale (internal DCN) and large-scale over the Internet (inter-DCNs or ISPs). We follow the same principle to find important applications across these two scales.

Applications may require different methods to expose their behaviour. For example, large-scale applications have limited visibility of the traffic as we cannot capture all the related traffic on the Internet. So, we need to understand the content popularity to obtain an estimation of the demand for content delivery. However, with full visibility of data centre scale applications, packets can be captured from the entire network to analyse the application behaviour. We first carefully select the applications, review related analysis methods, and then use specific methods to profile the applications.

1.2.1 Large Scale Distributed Applications

Large-scale applications are commonly experienced on the Internet, such as online searching [17], social network services [18, 19], and online video streaming [20, 21]. Of these, video traffic delivery is the most challenging due to the requirements of high throughput and low latency. With the widespread deployment of broadband access, video has become a key content type transported over the Internet. Thus, video-on-demand (VOD) services such as like Netflix, user-generated VOD services such as YouTube, and online video streaming services such as PPLive are all well studied in terms of content popularity [22–24], video delivery methods [25–29], and infrastructure [30–34].

Recently, a new type of video service on the Internet, *user generated live* video streaming, has become a significant component of Internet traffic [35], represented by applications such as Twitch [36], YouTube Gaming [37], Facebook Live, Periscope [38], and Meerkat. Of these, Twitch and YouTube Gaming are video-gaming-oriented streaming platforms. Video games are a form of entertainment enjoyed by a diverse, worldwide consumer base, and with such an exciting new generation of technology and content, the video game industry is expected to continue to thrive. In 2014, 51% of U.S. households owned a dedicated game console, and gaming is simultaneously expanding to additional devices, such as smartphones, tablets, and smart TVs, as many people are shifting their time from traditional TV or cinema into this more interactive form of entertainment [39]. This $93 billion industry is proliferating with an 8% annual growth rate similar to the

film industry [40, 41]. Traditionally, gaming has been a pastime enjoyed by those who choose to play. However, a trend emerged recently by which gaming has become an entertainment option for those who wish to consume passively. Major eSports tournaments are enjoyed by millions earning huge revenues (20% of the total game industry [40]), yet mainstream media outlets have paid little attention to such phenomenon. Video gaming content plus user-generated live video streaming created the undeniable popularity of Twitch, which recently was reported as the 4^{th} largest website in the U.S. by peak traffic [35], and the concurrent viewership exceeded 2 million users in real-time during a single event in early 2015 [42]. Though others have started similar services, (e.g., YouTube Gaming), they are yet to experience the same level of demand of Twitch [43, 44]. Its massive scale drew the attention of other prominent companies as rumours of Googles acquisition interests began to circulate around 2013, and in August 2014, Amazon purchased Twitch for $970 million. However, the user behaviour of such a popular platform remains poorly understood.

Twitch introduces at least two core innovations in presenting the video gaming content: (*i*) any user can provide a personal live stream (potentially to millions of viewers) and (*ii*) this upload must occur in real-time due to the live social interaction between consumers and producers. Twitch is oriented towards video games allowing users to broadcast their gameplay as well as to watch major eSports tournaments with professional players. Popular types of channels include amateurs broadcasting their gameplay, competitive eSports tournaments with commentaries, coaching game sessions, charity marathon events, and experimental large-scale cooperative events where games are played collectively. This user-generated live video streaming challenges the current video content delivery architecture. However, we do not possess much knowledge of how the infrastructure should be built.

In this thesis, we analyse the Twitch platform. Similar to previous studies on large-scale video services, we focus on user behaviour and infrastructure. To perform data-driven research, we collected data from the Twitch API to investigate the user behaviour

exposed by stream popularity. We then explored channel and game popularity, highlighting the nature of broadcasters as well as various unusual events observed. We performed a large-scale active measurement over hundreds of open proxies to probe the servers hosting the live stream. We discovered the footprint and streaming strategy of Twitch as a state-of-the-art content delivery strategy for user-generated live video streaming.

1.2.2 Data Centre Scale Distributed Applications

Apart from applications running at Internet-scale, we also profile an application running inside a data centre. A typical class of applications running inside data centres is distributed applications that enable Big Data processing. Today, data volumes are growing rapidly, which are generated from sources as diverse as mobile phones to the traffic and wireless sensor networks of many cities. By analysing datasets with significant volume, variety, velocity, and variability, we can discover more correlations to guide scientific research or business intelligence.

However, Big Data challenges include the capturing, storage, and processing of data. Relational database management systems and desktop statistics and visualisation packages often have difficulty handling an extreme amount of data. The work requires massively parallel software running on a large number of servers [45]. Previously, Big Data analysis platforms were built by corporations with a special need. For example, the first systems to store and analyse terabytes of data, named DBC 1012, was built by the Teradata Corporation in 1984 [46] and the HPCC (High-Performance Computing Cluster) system [47] was built by LexisNexis Group [48] in 2008. Following the same concept of having separate data storage and processing, Google published two papers on a scalable distributed file system, GFS (Google File System) [49], and a parallel-processing framework, MapReduce [50]. Both frameworks were very successful, so others replicated the approach, and an Apache open-source project named Hadoop adopted the implementation. Hadoop made Big Data analysis accessible to the public with heterogeneous and inexpensive commodity hardware. Based on Hadoop, the community went

on to enrich the functions by published a series of tools including stream processing, in-memory processing, NoSQL databases, machine learning suits, and cluster management components. These tools comprised a mature Big Data analysis eco-system, which then boosted Hadoops popularity by making it the industry standard for distributed data processing. As of now, most modern Big Data processing systems continue to use Hadoop for these sorts of applications making it the most fundamental tool in the community. As the leading Big Data analysis platform, Hadoop is expected to grow to a $50.2 billion market by 2020 [51].

Distributed systems like Hadoop heavily rely on the network for coordinating resources and tasks. Even though networks are essential for building any Hadoop cluster, the networking-related behaviour is not fully explored. Previous studies established the importance of networking, for example, Jiang *et al.* [52] listed I/O as one of the five most important factors that affect Hadoop performance (the other factors include data indexing, record parsing, data transformation, and block-level scheduling). Sur *et al.* [53] found that an increase of bandwidth in a Hadoop cluster from 1 gigabit to 10 gigabits can reduce the time for a sort operation by 11%. However, despite evidence that the network can improve the performance of Hadoop, most of the network-related studies are focused on flexible data placement to reduce the traffic when importing data into Hadoop [54–57]. Carefully placing the data blocks can indeed reduce the network traffic generated by Hadoop. However, this is only the beginning of data processing. A significant portion of the traffic is generated during processing rather than at the beginning, for example, as in shuffle traffic. We argue that a proper study of Hadoop network behaviour is necessary to deliver Hadoop traffic better. While we find previous studies on data centre networking, many do not apply to Hadoop. As indicated in [14], the Hadoop traffic pattern is different from other data centre applications, such as web or caching. The traffic patterns will result in different networking performance [1], overthrow existing knowledge on better support of such application. Thus, a critical problem to research in this study is filling the gap between Hadoop traffic with existing network

Table 1-A: A comparison of various data centre-related applications.

	YouTube	Netflix	**Twitch**	Web	OpenStack	**Hadoop**
Scale	large	large	large	large	local	local
Traffic pattern	C2S	C2S	C2S	C2S	S2S	S2S
User Diversity	low	low	low	low	high	high
Traffic Visibility	low	low	low	low	high	high
Traffic Demand	1b [58]	83m [59]	100m [60]	varies	varies	varies

performance evaluation techniques, so that network researchers can understand how to better support distributed applications like Hadoop.

In this thesis, we analyse Hadoop traffic in various scenarios to discover Hadoop traffic patterns and how to evaluate the network performance for distributed applications.

1.2.3 Summary

Table. 1-A summarises the features of related applications, including scale (over the Internet or internal DCN), traffic patterns (client-to-server or server-to-server), user diversity (how traffic is served differently to users), traffic visibility (can the traffic be acquired), and traffic demand (daily usage). As an online video streaming platform, Twitch is dedicated to providing user-generated video game streaming for millions of users on the Internet. Although different content is streamed over each users connection, the HTTP-based web requests are similar. On the other hand, Hadoop is an open source data analysis platform working in a distributed manner. Analysis tasks are offloaded to multiple nodes to perform various parallel processing algorithms. Due to the demand for low packet loss and low latency in data analysis, Hadoop is normally deployed over a local network, which enables full visibility of the network traffic. After identifying the applications, the goal is to explore these from both a usage and a network perspective by looking at what are the application behaviours, how is the traffic generated, and how does the traffic profile affect the network. By profiling Hadoop and Twitch, this thesis extends the understanding of application behaviour to two other important areas, namely distributed applications and user-generated live video streaming.

1.3 Research Goals

To acquire a better understanding of distributed applications from the networking perspective, this thesis proposes the following research targets.

First, regarding the Twitch application:

- *Characterise Twitch to explore the user behaviour behind popularity to reveal the uniqueness of user-generated live video streaming.* Twitch traffic is only generated when the broadcasters are streaming, and viewers are watching. So, the content popularity indicates the traffic generated, and we collect Twitch stream popularity to characterise this traffic. Unlike traditional VOD platforms where popularity is correlated with the video content, we find the primary factor driving popularity in Twitch is the streamer, or the person producing the streaming content. The streamer is part of the content and represents more than just the game or games they play.

- *Explore the infrastructure deployment, hosting, and redirection strategies of Twitch to reveal the content delivery of live video streaming.* As a large-scale web service, Twitch.tv serves millions of users on the Internet. Unlike traditional VOD platforms, the content delivered on Twitch is live and non-storable. Thus, we can profile the content delivery infrastructure of Twitch. We find that Twitch is using a different CDN strategy compared to traditional VOD. Instead of using ISP caches, Twitch manages its own data centres on different continents. We also find that the performance of this deployment is limited by Twitchs Internet peering connections.

Second, regarding the Hadoop application:

- *Characterise and model the traffic generated by Hadoop, under various cluster settings, jobs, and job settings.* By characterising the Hadoop traffic under various scenarios, for example, block size, input split size, TeraSort, and PageRank, we find that Hadoop traffic varies over only a few parameters. With such parameters taken

into consideration, we can acquire a broader spectrum of Hadoop traffic patterns so that the traffic further reproduced is more authentic.

- *Generate the Hadoop traffic in a simulation environment, so that the network performance of different networking techniques can be evaluated.* On the one hand, profiling network performance for distributed applications has been difficult due to the cost of hardware and labour, for example, experiments with various network topologies require hundreds of devices and several days to set up. On the other hand, network performance, *e.g.,* throughput, latency, and packet loss, directly affects the distributed application performance. To bridge the gap, we present a tool that simplifies the process and enables the evaluation of network components with authentic Hadoop traffic in ns3.

1.4 Outline of the thesis

The thesis is structured as follows:

- Part I-**Introduction** outlines the work of the thesis.
- Part II-**Background and Related Work**:
  - When serving users on a global scale, services need to cope with long content delivery distance and uneven traffic demand across various regions. Thus, the infrastructure deployment needs to be carefully designed, including deploying multiple servers in different regions. **Chapter 2** introduces the related work on existing large-scale Internet web services, including the infrastructure deployment and traffic demand measured by content popularity. It further illustrates an emerging type of Internet web service, the user-generated live video streaming in Section 2.3. Compared to traditional VOD, live video streaming requires lower latency due to the need for real-time interaction between broadcaster and viewers. Section 2.3.3 overviews the existing studies

on delivering online video streaming traffic.

  - Due to the importance of networking in distributed applications, **Chapter 3** briefly introduces data centre networking and distributed processing applications. Unlike large-scale web services that usually run on the Internet, distributed processing applications normally run inside a data centre network due to the high traffic demand and high data integrity requirements. Thus, we first explain the network performance measures in data centre networks in Section 3.2.1 to identify the objects of our study. We also briefly introduce research done on data centre networking in Section 3.2.2 from the physical layer topologies to transport layer protocols. Finally, we introduce the distributed processing applications represented by the Hadoop eco-system and the work done on studying the impacts of networking in distributed processing applications. With this, we develop a background to support our further study.

- Part III-**Profiling a Large-scale Online Video Streaming Platform**:

  - **Chapter 4** investigates the most popular live video streaming platform, Twitch, as an instance of a large-scale user-generated live streaming platform. We characterise the system from the data provided by the Twitch API to understand what drives the popularity of each stream. By diving into games, broadcasters, and tournaments, we reveal the spectrum of how popularity works in Twitch.

  - **Chapter 5** further investigates the infrastructure deployment of Twitch. Unlike YouTube or Netflix, where video content can be cached in different locations to distribute the load, live video streaming in Twitch is uploaded by broadcasters in real-time and delivered to viewers in real-time to enable interactions. Thus, we use proxies all over the world as vantage points to profile how Twitch offloads the content on a global scale.

- Part IV **Profiling a data centre Scale Distributed Processing Application**:
  - **Chapter 6** investigates Hadoop traffic as an example of data centre-scale applications. Little work has been done on the network aspects of Hadoop, so we start with characterising the Hadoop traffic generated under different scenarios, including cluster settings, jobs, and job settings. Based on the traffic observed, we propose a set of system level parameters to describe and fit the Hadoop traffic.
  - Based on the traffic fitting results, **Chapter 7** develops a simulation tool (called Keddah) that can reproduce the Hadoop traffic. We illustrate the specific design of Keddah to show how it imitates the Hadoop behaviour, including resource management, container-based processing, and traffic generation. We also compare the traffic from the simulation against the real traffic captured for validation of the simulation. Finally, we demonstrate a few examples that use Keddah to profile network components, including data centre topology and transport layer protocols.
- Part V-**Conclusion and future work**.
  - **Chapter 8** concludes the thesis by summarising the work and reviewing contributions along with comments regarding the limitations of this work and future directions for related networking innovations.

Part II

Background & Related Work

Chapter 2

Large-scale Live video streaming

Before we start profiling Twitch, we first look at the existing studies on large-scale video applications. Various services can be found on the Internet providing daily access to multiple types of media including text, music and videos. However, video is considered to be the most challenging application to network over the Internet due to the high bandwidth and low latency requirements.

In this chapter, we first look at the infrastructure deployment of various large-scale video services including YouTube, Netflix and Hulu. One important factor to justify the infrastructure is the traffic, which represented by content popularity [61–63]. Thus, we further look into the studies on popularity to help us justify infrastructure..

Even though multiple VOD, user-generated VOD and video streaming platforms have existed for years, an emerging type of video streaming service has become a major composition of Internet traffic recently [35]: user-generated live video streaming. In user-generated live video streaming, the video content needs to be transmitted with low latency, to enable live interaction between viewers and broadcasters. Later in this chapter, we look into the emergence of the platform Twitch and discuss the various techniques proposed to reduce the transmission delay in user-generated live video streaming.

2.1 Large-scale Video Services Infrastructure

To run a service at a global scale, the service provider needs to consider a set of problems such as support high traffic load and overcome the high transport latency. However, most of the problems can be solved by carefully choosing the infrastructure deployment. For example, high traffic load can be shared by multiple servers to reduce the load on each server; long delivery distance can be optimised by placing multiple servers in the Content Delivery Network (CDN) [64]. Thus, the infrastructure deployment is one of the most important topics in large scale video services. In this section, we will go through various architectures available for video services.

2.1.1 Background on Internet Topology

A detailed understanding of the Internets topology is essential for evaluating the performance of networking protocols and developing improved designs for resource provisioning [65]. There are three different levels at which to describe the network topology: the link layer topology, the network layer topology, and the overlay topology [66]. Link layer topology [67] is relatively difficult to profile, due to the sensitiveness of connectivity information and difficult in measurement.

However, there is considerable scientific literature devoted to the discovery of network layer topology. The network layer topology, sometimes referred to generically as the internet topology, is the key measurement of Internet structure. The Internet topology can be represented in four levels: IP interface level [68, 69], the router level [70, 71], Point of Presence (PoP) level [72, 73], and Autonomous System (AS) level [74–77]. During those studies, many important attributes are found, for example, the Internet topologies exhibit power laws for outdegree of node (domain or router) versus rank, number of nodes versus outdegree, number of node pairs within a neighbourhood versus neighborhood size (in hops), and eigenvalues of the adjacency matrix versus rank [78–80].

In this thesis, the Internet topology is seen as the foundation of application network. We focus more on the application network itself, or the overlay network composed of the application. There is a notable number of studies made on this area [81, 82], however, we will introduce a few that related to the network of video content delivery.

2.1.2 VOD Infrastructure

In video-on-demand platforms, the video contents are all uploaded and managed by the platform. Thus the infrastructure used is also relatively more straightforward. Use Netflix for example, Adhikari *et al.* [32] firstly identified the CDN locations used by three popular VOD services: Hulu, Netflix, and Amazon VOD. Then, Adhikari *et al.* [27] also found Netflix performs the server selection statically for each user, based on the user profile back in the year 2012. Not just Netflix, Amazon VOD uses only Akamai for content delivery; hence the selection policy relies on that of Akamai's. More recently, the specific architecture of Netflix CDNs are identified: Böttger *et al.* [34] found Netflix delivers its content to end users through its own caches (called Open Connect Appliances) that are either deployed within different host networks or placed at critical IXPs. In summary, the architecture of a VoD system like Netflix is presented by Figure. 2.1 where servers are deployed in both ISPs and IXPs to serve users from the same location.

2.1.3 User-generated VOD Infrastructure

User-generated VOD platforms allow users to upload videos for sharing. Managing millions of videos and delivering them to the users can be very challenging. One of the most well-known user-generated VOD platforms is YouTube, and many studies have been conducted on it. Adhikari *et al.* [30] collected Netflow traffic within a tier-1 ISP to locate YouTube data centres. They found only a few data centres in the US were in charge of distributing the videos around the world. After two years, the same authors further used PlanetLab nodes to enumerate YouTube servers and geolocated them in

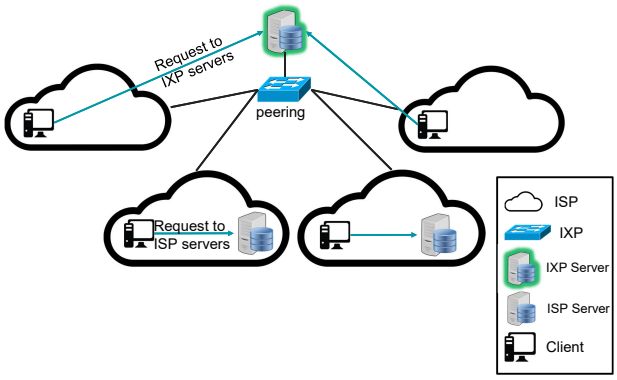

Figure 2.1: Netflix infrastructure architecture: servers are deployed in both ISPs and IXPs.

[83]. They successfully identified 47 different YouTube cache clusters over four continents from domain name service (DNS) records.

On a large scale, Adhikari *et al.* [31] used PlanetLab nodes to probe and measure YouTube, found YouTube is using many different cache servers hosted inside user ISPs. Torres *et al.* [26] captured traces in a campus network, showing the server selected by the YouTube CDN for streaming the video is usually the closest one to the user with few notable exceptions. Based on this, the architecture of YouTube can be captured in Figure. 2.2 where, servers are widely deployed in ISP caches with few exceptions from the company owned AS.

2.1.4 Live Video Streaming Infrastructure

Live video streaming is also one of the most popular video sharing services on the Internet. Company-owned videos are streamed in real-time to serve the contents in a television manner, thus named IPTV (Internet Protocol Television). One of the pioneer named WebTV (later acquired by Microsoft and renamed to MSN TV) accumulated as much as 800k users by 1999 [84, 85]. However, due to the high bandwidth required for the cen-

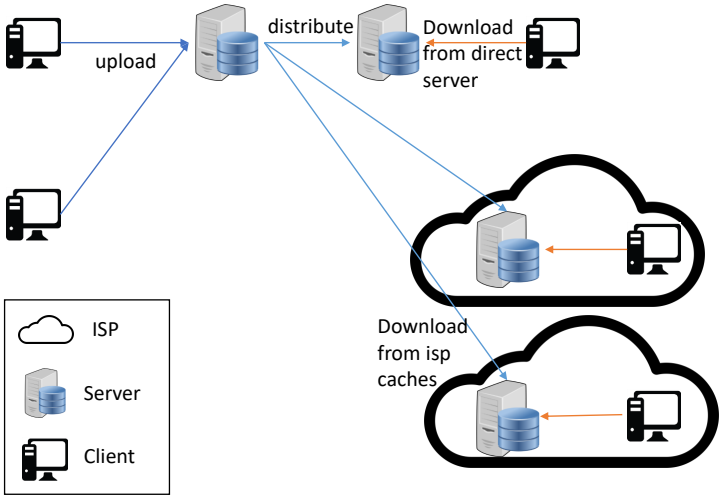

Figure 2.2: YouTube infrastructure architecture: servers are widely deployed in ISP caches with few exceptions from company owned AS.

tralized streaming system, researchers are often focused on bandwidth compression [86] or video encoding techniques such as Variable-bit-rate (VBR) [87] and scalable video coding [87].

Other the other hand, due to the same contents delivered among users, multicast and P2P (Peer to Peer) are often used to assist the delivery [88]. CoolStreaming [89], the first large-scale mesh-pull (pull videos from a mesh-shaped overlay peer network) live streaming system, demonstrated the feasibility to support a scalable video delivery service with reasonably good viewing quality over the best-effort Internet. PPLive is one of the most well-known live video streaming applications which, stands out due to its increasing popularity. Hei *et al.* developed active crawling apparatus to measure the global view of the PPLive network in [90], providing an understanding of how to architect a successful large-scale P2P IPTV system. More recently, three video streaming platforms including PPLive, SOPCast, and TVAnts are analyzed [91], with a systematical exploration of the mechanism driving the peer-selection in the different systems performed. While the P2P architecture can significantly reduce the traffic load from the

centralised streaming server, some unpopular channels can suffer from poor streaming performance due to lack of peers. Thus, like VOD or user-generated VOD systems, CDN based content delivery is also used in live video streaming [92].

In conclusion, by using a P2P fashion in delivering video streaming content as presented in Figure.2.3, the bandwidth cost of the company can be hugely reduced by offloading the traffic to the peers [93]. Also the latency between users can be ensured.

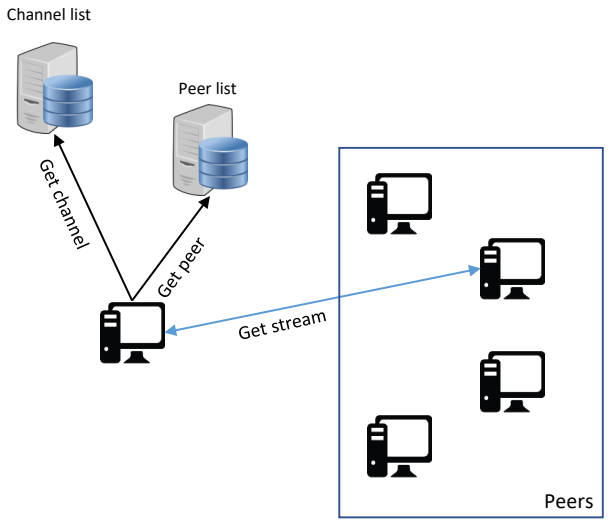

Figure 2.3: Due to the same content are shared among users, PPLive delivers traffic in a P2P manner to reduce the bandwidth and latency.

2.2 Popularity of Large-scale Video Services

User behaviour analysis is also an important part of the whole picture as the improvement of content delivery, or content management needs to be based on the understanding of end users [94]. Different methods have been developed to help understand behaviour. For example, mining browsing histories [95] and extracting information from online social media [96–98]. In this section, we focus on the measurements on content popularity as it is still the most obvious way of inferring traffic.

2.2.1 VOD User Behaviour

Krishnappa *et al.* [99] profiled 2300 videos from Hulu and found the content popularity follows a Zipf distribution and 77.69% of the content are repeatedly requested. Zeng *et al.* [100] found the content popularity is hugely affected by the content lifetime, for example, short lifetime contents on Digg are affected by recent popularity; however total popularity is a better factor in long lifetime content platforms such as Netflix and Movielens. Furthermore, content popularity is way more duplicated among users in regional VOD services such as BBC iPlayer. Karamshuk *et al.* [101] found as much as 88% of the traffic is duplicated among users. This concentrated content popularity explains the straightforward infrastructure deployment well: as long as few popular items are distributed to the cache servers, the viewing performance can be guaranteed.

Where over 70% to 80% percent of the video requests are multiple requests for the same content in commercial VoD, the same figure in the user-generated VoD platform YouTube is only 25% [102]. A more detailed comparison is made in [103], including content production rate, distinct publishers, length of videos, popularity distribution and use of caching. The nature of the UGC popularity shapes the popularity distribution, including generates a long tail, alters the skewness and breaks the power-law behaviour for very popular contents. Caching the most popular contents can offload the server traffic by as much as 50%, way fewer than the VoD.

Beyond YouTube, a more extreme example showing how UGC content differs from traditional VoD is presented in [104]. Tyson *et al.* found the content popularity on YouPorn is not related to duration or category, but affected by upload age, number of categories and accessibility. New video content can trigger a short burst of activity in general, then followed by a popularity collapse.

All these characteristics of UGC indicate the difficulty in predicting and managing video content. Thus, more caches need to be deployed infrastructure wise thus the popular contents can be rapidly replicated to ensure the viewing experience as well as

reduce the over all traffic. This can further justify the infrastructure architecture we discussed before.

2.2.2 Live Video Streaming User Behaviour

Ali *et al.* [105] study the traffic characteristics of controlled peers on PPLive and SopCast, found a Weibull distribution of peer lifetime. Passive sniffing was also utilized to study the traffic pattern of PPLive, PPStream, TVAnts, and SopCast in [106]. Also, a few works measured the global popularity of channels. Li *et al.* [22] found that PPLive provides different levels of QoS support for various channels: popular channels often have sufficient resources for smooth playback, while unpopular channels experiencing serious issues such as occasional glitches, re-buffering and broken streams. That said, even though use of peers in live video streaming can help reduce the traffic replication, caching servers are still needed to help ensure the performance of less popular channels.

2.3 User-generated Live Video Streaming

Recently, a new type of video services on the Internet, which is *user-generated live video streaming*, has dominated Internet traffic [35], represented by applications like Twitch [36], YouTube Gaming [37], Facebook Live [107] and Periscope [38]. Deliver user-generated live video streaming can be very challenging due to (*i*)the user-generated part will make the content popularity difficult to predict and manage; and (*ii*)live video streams need to be delivered from streamer to viewer in time to enabling real-time interaction. Interestingly, both Twitch and YouTube Gaming are video gaming oriented streaming platforms and have accumulated millions of users. In this section, we first shed light in online gaming and current studies on Twitch and YouTube gaming.

2.3.1 Online Gaming and Twitch.tv

Video gaming has a history spanning decades. Early research focused on the impact of gaming on both society and individuals. For example, much work has investigated the effect of gaming on school children [108], teenagers [109] and even adults [110, 111]. Recently, researchers have turned their attention to online gaming, looking at its evolution [112]. Furthermore, online social gaming has recently emerged as a hot topic, as it integrates the fields of gaming with that of social networking [113–115]. That said, the popularity of Twitch and YouTube Gaming share a popularity model similar to any other physical sports. This makes Twitch popularity different from existing UGC platforms, with more similarity shared between IPTV platforms.

Twitch is video game oriented live video streaming platform. Users can broadcast their gameplay, as well as watch large eSports tournaments between professional players. Popular types of channels include amateurs broadcasting their gameplay, competitive eSports tournaments with commentaries, coaching game sessions, charity marathon events, and, finally, experimental large-scale cooperative events where games are played collectively. Users in Twitch can take one of two roles, broadcaster or viewer. A broadcaster is somebody who streams their game play via a dedicated channel, whilst a viewer is somebody who watches the channel. Each streamer is limited to one live channel, which is only online for a fixed period of time when the player is broadcasting. The user interface of the live channel is shown in Figure. 2.4, including the streamer, the gameplay, the title of the channel, name of the game, number of concurrent viewers and number of accumulated viewers. To facilitate communication, the channels are enabled with an in-built chat room to allow both the broadcasters and the viewers to interact with each other as shown in the right part of Figure. 2.4.

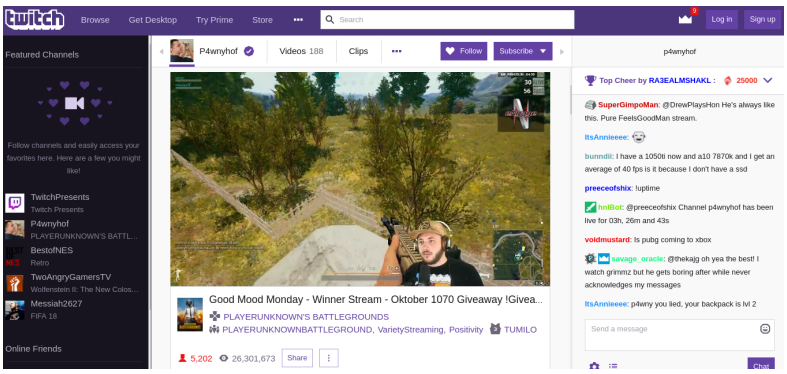

Figure 2.4: Screenshot of Twitch.tv, a platform for users to share their game playing video streaming worldwide.

2.3.2 Popularity of User-Generated Live Streaming

A few other researchers have started to look into this new game broadcast platform from different angles, including how live broadcasting has shaped the way people treat video games [116]. A particular case is "TwitchPlaysPokemon" which, by enabling viewers playing the game "Pokemon" together using commands from the chat room, has accumulated over 1 million viewers on Twitch [117]. This viral channel can boost Twitch popularity and also attract more than 80% of Twitch viewers into this solo channel. Such unpredictable behaviour which, in terms of both popularity and lifttime, makes Twitch a different platform compared to traditional UGC platforms such as YouTube [118]. Following that, more attention has been put on the content delivery side, including traffic delivery over the Internet [119, 120] and measurement of the live streaming community [121]. In general, the key to measuring the traffic or community activity is going to be the *popularity*, which is considered by viewing figures in the Twitch. A few works trying to characterise the popularity in Twitch, for example, the first look at Twitch popularity with two months of dataset [122] shows the popularity distribution and duration; a further look pointed out the popularity of channels is highly predictable by the channel's early viewing figures [123]. However, none of the works have investigated Twitch's popularity in depth, for example, study the popularity in the long term or reveal the correlation between popularity with contents. We learned from VOD and UGC that

contents have a huge impact on the user behaviour and popularity, thus similar principle could apply to user-generated live video streaming as well. Thus understand the user behaviour first is important before evaluate the underline infrastructures.

2.3.3 Infrastructure of User-Generated Live Streaming

As a video streaming service explosively growing recently, the content delivery techniques for user-generated online video streaming platforms like Twitch has been discussed in the research community. Claypool *et al.* [124] shows Twitch is the favourite video streaming platform for e-sport fans watching live streams of video games, and the channel popularity is predictable by its early stage. Implementation wise, Shea *et al.* [125] evaluated the hardware costs for stream broadcasting, and found that general hardware encoders can greatly reduce the energy usage without affecting the performance. In terms of transportation, a preliminary design of streaming delivery using peer-to-peer approach is proposed in [126]. And Chen *et al.* [127] evaluated a prototype of a live streaming platform built with Amazon Cloud/Microsoft Azure and PlanetLab nodes.

Even though many different techniques have been proposed, none of them has investigated the current infrastructure deployment of Twitch. With the production, consumption, and delivery of the video content in real-time, the infrastructure needed will be different from traditional video platforms. However, without the current deployment of Twitch, there is no way to compare how other content delivery techniques can better perform.

2.4 Summary

The history of delivering video on the Internet is almost as long as the history of the Internet. By years of development, video delivering has interacted with nearly every aspect of the network: from queue and Quality of Service (QoS) on link layer, Real-time

transport protocol (RTP) and Real Time Streaming Protocol (RTSP) on network layer 3, to various video coding techniques on presentation layer and different streaming techniques on application layer. As the purpose of this thesis is to study the user behaviour and underlay architecture of a user-generated live streaming platform, we show some of the recent and relevant platforms as a reference to our study.

In this chapter, we have reviewed the previous studies on the infrastructure and popularity of existing large-scale video platforms including video-on-demand, UGC, and live streaming. However, a new type of video sharing service named user-generated live video streaming may overthrow the knowledge acquired from those existing platforms. Although previous work has focused on platforms in which static (*i.e.,* non-live) content is being delivered, Twitch suffers from far greater constraints due to its live real-time nature (making caching redundant). As a platform serving user-generated live video streams, Twitch will be different from other traditional VOD platforms. We found a few requirements that distinguish Twitch from the previous video streaming platforms: (*i*) Any user can provide a personal live stream (potentially to millions of viewers); (*ii*) This upload must occur in real-time due to the live social interaction between consumers and producers around the globe. These core innovations could differ the user behaviour of Twitch users from existing video sharing platforms like YouTube and Netflix. The difference in viewing behaviour and the nature of live content will then ask for a different content delivery architecture. Hence, realistic performance measurements of *user generated* live video streaming platforms very valuable yet still missing.

Chapter 3

Distributed Processing Applications and Data Centre Networking

Distributed systems are not only seen on the Internet scale but also regularly running inside data centres. Recently, the BigData trend drives the blooming of large volume data processing applications, often a distributed processing application. In this chapter, we look at the related studies on distributed processing applications and how networking impacts the processing performance.

Mainly due to the intensive data transmission and low packet loss tolerance, the distributed processing applications usually run inside a data centre. A data centre is a collection of computational, storage and network resources interconnected by a communication network [128]. Although very limited studies can be found on improving the networking performance of distributed processing applications, data center networking is one of the topic that devoted to improve data center application performances. In the second part of this chapter, we briefly look at data center networking studies. We first show the metrics that used to measure the networking performance, so we know

what performance means. After that, we look at the previous works that aim to improve network performance for specific applications in data centre.

3.1 Distributed Processing Applications

Distributed processing applications are commonly used in the data centre in the era of "BigData". When analyzing data with a huge volume, a single machine is no longer an option for data processing. Due to the massive data volume, low transmission latency and high data integrity requirements, the distributed processing applications are usually running inside a single data centre. In this section, we will introduce the distributed processing application, represented by the Hadoop eco-system. One of the methods to evaluate the distributed processing application performance is through benchmarking. However, to simplify the process, many simulators are proposed. We will give a few benchmarking and simulator examples, and finally discuss the importance of networking in distributed processing.

3.1.1 Hadoop And The Eco-system

Big data challenges the traditional data analysis process from capture, store to process of data. Relational database management systems and desktop statistics- and visualisation-packages often have difficulty handling big data. The work requires massively parallel software running on tens, hundreds, or even thousands of servers [45]. Previously, big data analysis platforms often built by corporations with a special need. Teradata Corporation marketed the parallel processing DBC 1012 system back in 1984 [46] that, was the first to store and analyse terabytes of data in 1992. In 2000, Seisint Inc.(now LexisNexis Group [48]) developed a C++ based distributed file-sharing framework for data storage and query. After acquiring ChoicePoint, Inc. and their high-speed parallel processing platform, the two platforms were merged into HPCC (High-Performance Computing Cluster) system [47] in 2008. In the meantime, Google published two papers

on a scalable distributed file system GFS (Google File System) [49] and a parallel processing framework MapReduce [50]. On the one hand, GFS provided fault tolerance while running on inexpensive commodity hardware, and delivered high aggregate performance to a large number of clients. On the other hand, MapReduce split and distributed queries across parallel nodes (the Map step) then gathered and delivered results (the Reduce step). Both frameworks were very successful, so others replicated the algorithm, and the implementation was adopted by an Apache open-source project named Hadoop which, contains a GFS-like storage implementation named Hadoop Distributed File System (HDFS) and a computation framework implementation named MapReduce.

Under the Hadoop framework, large datasets can be processed in a distributed manner using HDFS and MapReduce. After its success, more components were proposed to complete the ecosystem. Some of the early applications are Hive [129] (for integration with traditional DBMS), Pig [130] (as a high-level language translating between SQL and MapReduce), HBase [131] (as a NoSQL style key-value database on HDFS), ZooKeeper [132] (for coordinating the configurations on the cluster) and Yarn [133] (for MapReduce job scheduling). These applications still focus on data storage and management, including accessing data from platforms other than HDFS or storing SQL-like data structures in HDFS. Soon enough, applications aim to enhanced processing functionalities have been developed, including Giraph [134] for graph processing, Mahout [135] for machine learning, Sqoop [136] for data transport between the database with Flume [137] for log style data collecting. These applications provide another abstraction layer on top of MapReduce so that complicated algorithms can be easily achieved without strictly using the mapper and the reducer. To further complete MapReduce, different processing schemes are proposed including in-memory processing with Spark [138], stream processing with Storm [139] and complicated job priority arrangements with Tez [140].

A more specific list of Hadoop related projects can be found in [141], and we show some of the most commonly used ones in Figure. 3.1 which, contains components from data storage to data management. Although a set of processing schemes are available

with Hadoop, MapReduce is still the most fundamental and commonly used one. Many other processing schemes still share the same processing paradigm as MapReduce, for example read/write data from/to HDFS and shuffle intermediate results between different nodes. Thus, to understand the network traffic of Hadoop, we still start from investigating MapReduce. By studying the network behaviour of MapReduce, we can have a good coverage on the networking behaviour of distributed processing applications.

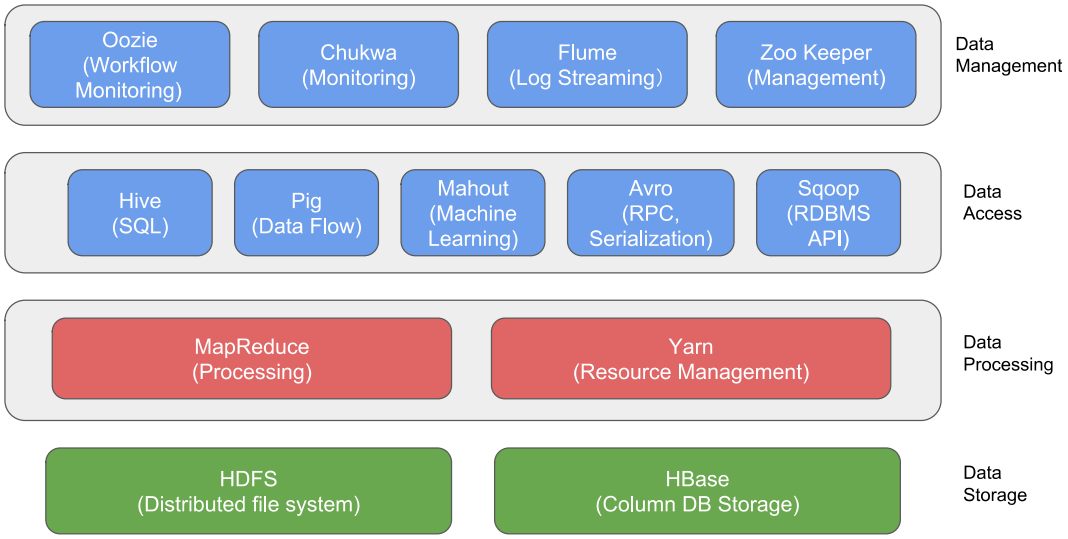

Figure 3.1: The Hadoop eco-system, consisting data storage, data processing, data access, and data management components.

3.1.2 How Hadoop Works

Hadoop MapReduce [50] is a system designed to support automated parallel data processing across multiple compute servers. It consists of a distributed storage system HDFS, and a distributed execution framework called YARN [133]. MapReduce defines a two stage execution flow where a mapping process is followed by a reduction process. Both mapper and reducer tasks are allocated to Java Virtual Machine (JVM) containers dur-

ing task assignment. Jobs start with the YARN Resource Manager allocating containers to nodes. If jobs require input data that needs to be processed by mappers, the allocation will try to assign containers onto nodes where the data split is stored locally (in the HDFS). Figure 3.2 presents the structure of a complete Hadoop job containing multiple mappers and reducers. Map containers read the appropriate data from HDFS (traffic identified by TCP source port 50010). Once each mapper has retrieved its allocated data blocks, it begins to execute the computation (*e.g.,* sorting). Results are partitioned by key, with reducers aggregating all emitted values belonging to the same key. Thus, results from mappers are then transferred during the shuffle step to the reducer containers (traffic identified by TCP source port 13562). Each reducer node computes results for each key, and once all computation has been completed, the reducer writes the result back to HDFS (traffic identified by TCP destination port 50010). Although the reducer nodes are often on the same node of mapper, data exchange is still required to deliver the mapper results to the reducer. This means that the Hadoop application generates a number of traffic components during job execution.

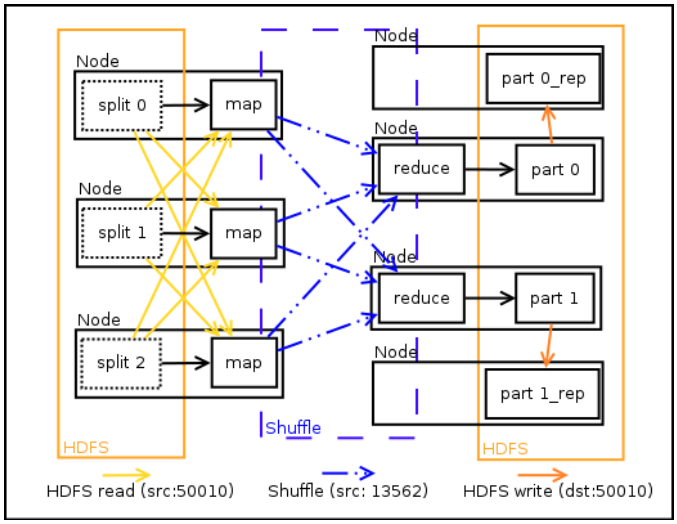

Figure 3.2: MapReduce data processing paradigm.

3.1.3 Networking Impact on Distributed Applications

Based on the way Hadoop works as explained in the section above, the network plays a very important role in functioning Hadoop. With the need of combining computation and memory resources from multiple machines, the network is essential for distributed systems to work. However, just a small number of studies have investigated the network behaviour of mainstream "Big Data" platforms such as Hadoop. For instance, [142] discusses the performance of Spark, finding that job completion time can be reduced by only 2% through network improvements. A more recent article [143] pointed out that these results are partially an artefact of specific platform optimisations of Spark (namely, heavy data replication to avoid network traffic). In contrast, they found that PageRank jobs can be improved by up to 3x by increasing the network capacity from 1 Gbps to 10 Gbps.

The lack of studies on the networking aspect of distributed processing application does not mean the network is not important. Recently, people realise the network not only affects the distribution processing by delivering traffic passively, but also can help achieve a better processing performance by interfering with the traffic. In-network processing tools such as NETAGG provides an on-path aggregation service that transparently intercepts aggregation flows at edge servers using shim layers and redirects them to aggregation nodes [144]. With NETAGG deployed, the time spend on shuffle and reduce can be hugely shorten compared to typical data centre set-ups. These works have confirmed the vital role of the network in the performance of distributed processing platforms such as Hadoop MapReduce. However, in-network processing can only help with aggregation type of operations so far. Better utilize of network requires a deep understanding of Hadoop traffic. Thus, we aim to profile the network traffic behaviour of Hadoop so that the knowledge can benefit both passive network transport and active in-network processing.

Although the network plays such an important role in functioning Hadoop, the net-

work related behaviour of Hadoop is still poorly understood. For example, we still do know the traffic pattern of Hadoop, no to say how the traffic is going to be varied based on different jobs. This lack of understanding prevents researchers from proposing innovative network techniques on better support of Hadoop traffic delivering, thus enhance application performance from the network perspective.

3.1.4 Distributed Application Performance Benchmarking

To address a common standard in measuring processing performance, benchmarking is widely used. By running a certain set of jobs with similar datasets, different applications can be put together for comparison. Various studies have looked at Hadoop workloads, primarily for cluster benchmarking (*e.g.,* comparing job completion times across clusters). Hadoop provides some benchmarking tools. TeraSort [145], later standardised as TPCx-HS [146], is the most popular benchmarking job for cluster comparison. There are also more diverse Hadoop benchmarking projects such as the Big Data benchmark [147] and HiBench [148]. These provide a more diverse set of Big Data processing use-cases, such as machine learning or graph processing (*e.g.,* HiBench uses Pegasus [149] to benchmark PageRank).

In our work, we aim to extend the purpose of benchmarking jobs from just computation performance evaluation to generating standard Hadoop job traffic. As benchmarking jobs are carefully selected to expose the Hadoop performance, we use those jobs to expose the network traffic generated during processing.

3.1.5 Simulating Distributed Application Performance

While evaluating the distributed processing application via benchmarking is made possible, setting up the cluster and running various jobs is still time and resource consuming. Thus, many simulation tools are proposed to reduce time and cost spent on benchmarking, and get an estimate from the simulation. Many processes can be simulated now,

including task prediction [150, 151], computation resources simulation(CPU, memory, storage [152]) and job complete time prediction [153, 154].

In a wider area, the simulation approach has been long used when studying the performance of distributed systems [151, 155]. Although some work has addressed the importance of networking in distributed processing systems [155–157], network related simulation has not been proposed until very recently. In 2016, researchers have developed a traffic modelling tools for Hadoop, including a C++ based Hadoop traffic simulation tool [158] and a Docker based Hadoop cluster with mininet defined topology to generate the Hadoop traffic [157]. However, these tools do not provide any flexibility in reproduce the Hadoop traffic with existing network protocol stack. Thus, the tool for investigating network performance with Hadoop traffic is still missing, for example traffic characterisation, performance evaluation, and network resource management for distributed processing applications.

3.2 Data Center Networking

While we aim to evaluate the network performance of Hadoop, we can use data center networking studies as a background. As Hadoop is mainly deployed in the data center in practice, we can still learn from many previous data center networking techniques. Data centers often need to contain tens or even hundreds of thousands of server resources and expected to be scalable and efficient to contain more [159]. Many works have been devoted to understand as well as to improve the data center networking performance. In this section, we first explain the metrics commonly used for measuring network performances, and then we show the studies made to improve networking performance.

3.2.1 Network Performance Metrics

3.2.1.1 Throughput

Throughput is the number of packets successfully delivered per unit time, measured in bits per second (bps) [160, 161]. Throughput can intuitively show the available bandwidth, as well as the hardware limitations. The *Time Window* is the period over which the throughput is measured, often dominate the calculation of throughput. Maximum network throughput equals the TCP window size divided by the round-trip time of communications data packets. However, there are many overheads accounted for in throughput in reality, including latency, TCP receive window size and system limitation, which means the throughput measured in reality is less than the theoretical value. Thus, the results of throughput are often measured by goodput which is the application-level throughput.

3.2.1.2 Latency

Latency is the time delay in packet transmission, often measured either one-way (the time from the source sending a packet to the destination receiving it), or round-trip delay time (the one-way latency plus from destination back to the source) [160, 161]. In packet switch networks, the latency consists of propagation delay, serialization delay, queuing delay and processing delays. Propagation is only affected by the physical distance between the source from the destination and calculated by the speed of light. The serialization delay is affected by the speed of the link, thus the time required to transmit the packets over. The counterpart in the network devices is processing delay, which consumed while a network device processing the packets. Propagation, serialization, and processing delays are relatively static depends on the hardware limitations. However, queuing delay occurs when a network device receives multiple packets from different sources heading towards the same destination. Typically, only one packet can

be transmitted at a time. Thus some of the packets must queue for transmission, adding an extra delay on the packet transmission.

3.2.1.3 Jitter

Jitter is defined as the variation in the delay of received packets [160, 161]. A large jitter won't necessarily cause trouble in packet switch networks, especially TCP based application can handle the jitter variance by nature. For example, TCP can adjust the retransmission timeout to gradually adapt to the jitter. However, when comes to UDP based applications such as VoIP (Voice over Internet Protocol), large jitter can cause poor communication performance by causing a glitch in sounds [162].

3.2.1.4 Error rate

The number of bit errors is the number of wrong bits received over a communication channel due to noise, interference, distortion or bit synchronisation errors [160, 161]. In the wired environment, the physical media is relatively more stable, and error is often caused by hardware design or conflicting in resource (*e.g.,* CSMA/CD).

3.2.1.5 Packet loss

Packet loss occurs when packets are travelling across a computer network and failed to reach their destination [160, 161]. Packet loss is measured as a percentage of packets lost against packets sent. In video streaming and online game applications, packet loss can affect the user experience. Many factors can cause packet loss, including link congestion, faulty networking hardware, packet drop attack.

3.2.2 Improving Network Performance

With the measurement of network performance established, lots of works have been done on improving the network performance. In this thesis, we will discuss how network performance can be improved for distributed processing applications. Thus, in this section, we first briefly look at common approaches done on improving the network performance, ordered by the OSI 7-layer model.

3.2.2.1 L1-DCN Architectures

For almost 20 years, the primary data centre network design has been the three-tier approach of the core (L3), aggregation (L2/L3), and access (L2) [163, 164]. Drawbacks have been exposed over the years with different requirements on the traffic delivery: design was primarily vertical (servers to outside) rather than horizontal (between servers) [165] which leads to an arbitrary L2/L3 boundary. Also because of high oversubscription rate of the core/aggregation layer, poor reliability, low bandwidth utilisation and a painful process of moving servers across subnets, the three-tier design is no longer suitable for today's fast-changing, innovation-driven data centre. Various technologies are proposed to bring flexibility to data centre networking, such as software-defined networking (SDN) [166–168] and network virtualization [169, 170]. Users can deploy functions into the network in an on-demand manner, dynamically change the network configuration based on traffic.

Over the last decade, the research community has put a lot of effort to avoid the drawbacks of tree-based topologies by exploring other types of architecture, such as:

1. FatTree like topologies, including an architecture that leverages commodity Ethernet switches to deliver scalable bandwidth for large-scale clusters such as FatTree [171] (Figure. 3.3 shows a FatTree instance with n=4) and a 40G datacentre-scale fabric called Jupiter [172];

2. Clos like topologies: data centre fabric from Facebook [173] and VL2 [16], an architecture providing uniformly high end-to-end capacity;

3. Torus like topologies, including BCube, an architecture specifically designed for shipping-container based, modular data centres [174], MDCube, mega-data centres topology [175], and a 3D Torus topology called CamCube [176];

4. Hybrid DCN architectures, including Dcell, a recursively defined structure without single point failure [177] (Figure. 3.4 shows a two layer Dcell topology with n=4) and it's derivative FiConn [178];

5. Star topology like a scale-free architecture Scafida [179]

6. Random topology like Jellyfish [180].

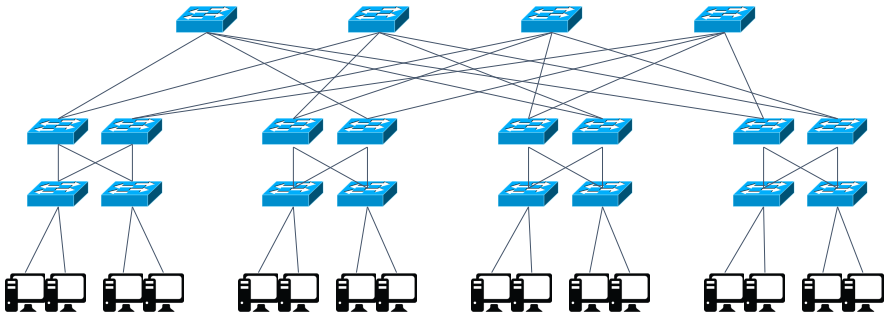

Figure 3.3: A FatTree topology with k=4.

In terms of the deployment, different DCN topologies are limited by the cost [181], number of wires [182] and energy efficiency [183]. Also, different topologies are compared on performance such as throughput [1], and reliability [184]. With so many topologies to choose from, it is very difficult to find the right one given an application. We will further address that in the case of Hadoop is part IV of this thesis.

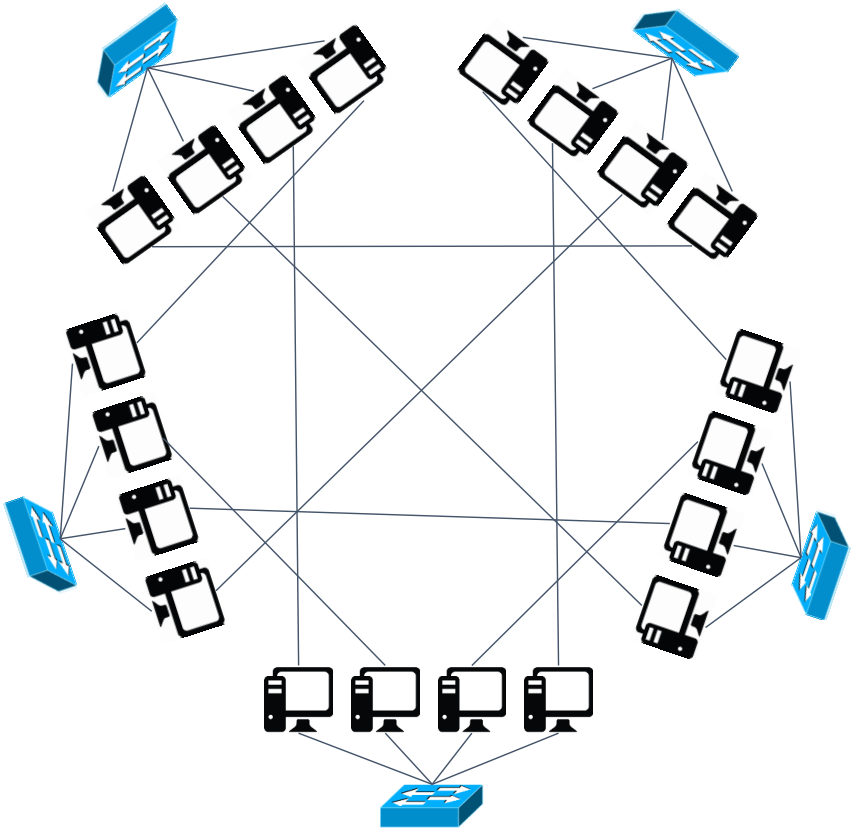

Figure 3.4: A Dcell topology when n=4.

3.2.2.2 L2-Active Queue Management and Scheduling

Active queue management (AQM) is the intelligent drop of network packets inside a buffer associated with a network interface controller (NIC), when that buffer becomes full or gets close to becoming full, often with the larger goal of reducing network congestion. In general, AQM consists of queue management and queue resource scheduling.

Queue management actively drops packets to avoid the congestion rather than drop all the packets when the queue is full. Representative works are Random Early Detection (RED) [185] which uses parameters to set when and how fast the packets are dropped. Many derivative solutions have been proposed based on RED. One of the most recent ones is Controlled Delay (CoDel) [186], which learns the parameters from the traffic and drops packets automatically. While just dropping the packets on the

interface is not enough, methods are proposed to notify the source to reduce the speed. Well-known methods like Explicit Congestion Notification (ECN) [187] and Random Early Marking (REM) [188] are used together with RED. For example, when the ECN flag is set by RED, the source will reduce the packet window to avoid further congestion.

On the other hand, queue resource scheduling can provide differentiated services (DiffServ) to traffic flows when the queue resource is limited. A well-known method is Priority Queue (PQ) [189] where priority flows are always served first, and Fair Queue (FQ) where flows are categorized as classes and each class gets a fair share. In FQ, more sophisticated methods are implemented such as serving one packet at a time Weighted Round Robin [190], and serving one class at a time Weighted Fair Queuing [191]. More recently, Virtual Queue [192] is also used together with REM to enable sending a notification back to the source to reduce the congestion actively.

In general, the use of AQM can reduce the latency and packet loss caused by congestion. However, similarly, with so many different techniques to choose from, only using the right one can help with the performance. However, the selection process is still unclear for distributed applications like Hadoop.

3.2.2.3 L4-Transport Layer Protocols

Traditionally, AQM can reduce the network congestion by scheduling queues on network interfaces. However, in DCN scenarios, AQM can not reduce end-to-end latency. Thus sophisticated layer four protocols like TCP and Stream Control Transmission Protocol (SCTP) are proposed.

The performance of TCP [193] is first studied in DCN, then all kinds of mutations are proposed to cope with different traffic instances, such as TCP-MT [194], multipath TCP [195], DCTCP [196] and D2TCP [197]. One of the problems found in DCN while using TCP is the performance collapse while multiple senders are communicating with a single receiver, named TCP incast [198]. Researchers have been working on congestion

control methods such as quantized congestion notification (QCN) [199], specifically with TCP incast [200], Approximate fairness QCN in multi-tenanted data centre [201], data centre QCN [202], congestion estimation using round trip time (RTT) [203, 204] and transmission window based congestion control [205].

Other than optimisation in protocols, flow scheduling is another method to increase DCN efficiency and utilisation. Hedera [206] proposed a flow scheduling framework for FatTree topologies. The central scheduler can outperform the hash-based Equal-cost multi-path (ECMP) load-balancing with the knowledge of active global flows. Recent work [207] further improved the performance to near-optimal coflow scheduling in DCN.

3.2.3 Conclusion

With so many networking techniques to choose from, it is very difficult to find the right one given an application. One simple approach would be, with those performance metrics listed in Section 3.2.1, we can directly profile those networking techniques to find the best-performing ones. Many approaches have been proposed to simplify the process, for example explore the performance in a simulation environment. To generate authentic network traffic in simulator, different methods are used including a single model in the early stages [208, 209], or more recently traffic generated by characterising the traffic from different scenarios [156, 210]. However, those monotonous traffic can not fully represent the network behaviour of sophisticated distributed applications such as Hadoop.

To fully expose the application behaviour to get a valid performance evaluation, the traffic used needs to be specifically studied from the application. For distributed applications like Hadoop, the behaviour is also affected by the jobs and processing parameters. Thus, none of the existing traffic patterns can be used to generate the Hadoop traffic. We will further address that in part IV of this thesis

3.3 Summary

This chapter has briefly introduced the data-centre specific networking research which, can provide a background for the network related study in this thesis. One of the claims this thesis makes is that with so many network topologies and protocols to choose from, it's very difficult to find the right one given a particular application. Network performance evaluation is a complicated process: (*i*) setup a physical testbed with a certain topology; (*ii*) switch from different protocols, which often requires compiling the operating system; (*iii*) setup the application to profile and finally (*iv*) repeat the first three steps with different topologies or protocols to find the best networking components. The whole process is time and resource consuming, even though better performance is not guaranteed in the end.

On the other hand, with the bulk of the studies of distributed processing applications focus on the application level, the networking has been ignored for the past years. However, this does not imply that the networking is not important to the distributed processing applications; on the contrary, the network is not only essential but also affects the performance of distributed processing. Thus, in part IV of this thesis, we will further look into the networking aspect of the distributed processing applications. The simulation approach that is widely used in this area inspired us to propose a network related simulator so that the performance evaluation for the distributed application can be easily done.

Part III

Profiling a Large-scale Online Video Streaming Platform

Chapter 4

Traffic and Popularity: A Case Study of a Large-Scale Web Application

4.1 Overview

Twitch is a live streaming video platform which allows users to broadcast themselves playing games to others. Streams include playthroughs of games by amateur users and large-scale broadcasts of eSports competitions. To facilitate communication, the channels are enabled with an inbuilt chat room to allow the users (both broadcasters and viewers) to interact with each other. Hence, user-generated live video streaming platforms like Twitch introduce two core innovations: (*i*) Any user can provide a personal live stream (potentially to millions of viewers), and (*ii*) This upload must occur in real-time due to the live social interaction between consumers and producers.

These core innovations could differ the user behaviour of Twitch users from existing video sharing platforms like YouTube and Netflix (Chapter 2). In this chapter, we

characterise the Twitch platform. We collect information on their temporal streaming patterns, as well as their popularity. Through this, we explore channel and game popularity, highlighting the nature of broadcasters, as well as various unusual events that can be observed. In summary, we made several key observations through this chapter:

- Section 4.2 briefly introduce the Twitch platform and provides insight into the growing scale of Twitch.
- Section 4.3 explores the popularity of different games against a set of game attributes
- Section 4.4 continues to explore the popularity from the channels perspective, due to game attributes do not explain Twitch popularity well.
- Section 4.5 concludes this chapter and highlights the next work.

4.2 Measurement Methodology

For data-driven research, we collect data from Twitch to conduct analysis. We first present a brief overview of the dataset, before moving on to describe our unique dataset collected over 11 months.

4.2.1 Dataset

To study Twitch, we have collected a dataset spanning 11 months from February 2014 to December 2014. This was collected using the public Twitch API, which we contacted repeatedly to extract all statistics available. Users in Twitch can take one of two roles: broadcaster or viewer. A *broadcaster* is somebody who streams their gameplay via a dedicated *channel*, whilst a *viewer* is somebody who watches the channel. Each streamer is limited to one live channel, which is only online for a fixed period of time when the player is broadcasting. For each channel, we retrieved the game being played, the number of viewers, the title of the channel and the broadcaster's name. We repeated

this every 15 minutes to build a time series of metadata for every channel in the system. Twitch has even become so popular that nearly **5k** streamers are trying to make a living by broadcasting [211], the subscription option can be seen as an identification of professional streamers. We collect all the subscription-based channels to justify the data. Furthermore, we also collect the YouTube and Twitter information from users' description in December 2014, which we followed to capture the YouTube subsection number and Twitter follower number of each streamer.

To investigate the effect of events in Twitch, we collected additional traces from the DreamHack winter tournament channels. DreamHack is the world's largest computer festival, and its trace could help us understand the behaviour of Twitch during events. An overview of the datasets is presented in Table 4-A, showing the information we collected, including the time interval between each API request, and how many samples were collected in total.

Data	**Sample Interval**	**#Samples**
All Channels	15 mins	323M
DreamHack Channels	5 mins	12731
All subscribable channels	n/a	4893
Streamer's Youtube info	n/a	78k
Streamer's Twitter info	n/a	135k

Table 4-A: Description of Twitch datasets.

4.2.2 Identifying Games

A fundamental question is what content is actually streamed on Twitch? To figure out that, we use the "game" field return from the Twitch API. We observed a total number of 127,497 different names from all the streams, which is a huge amount of games for the Twitch system. While having doubts about these numbers, we found the game name part of each stream can be manually filled by the streamer. We found many instances where the streamer mistypes the name of the game, the upper case, and lower case, or the space between words. These ambiguous names can easily double the number of real unique games.

To resolve the metadata into real game names, we compare the words from Twitch to an external games metadata. To search all the name of games existed, we first use "TheGamesDB.net" [212], an open, online database for video games, which contains 17,994 unique games. We matched those names from GamesDB against Twitch dataset and discovered 10k different names. But a manual checking of the results shows some of the older games have not been included in the GamesDB, and players can emulate the hardware to play on PCs. We further use another database "GiantBomb" [213], an American video game and wiki, to collect more game names. We got 50k games in total after combining the items from both 'GamesDB' and 'GiantBomb', and 12k of them are found in Twitch directly. Including names ignoring space and cases, we have 95% of the total occurrences of games appeared in Twitch covered.

4.2.3 Twitch Usage Growth

We begin our analysis by inspecting the overall trends of Twitch as a service.

There are two types of users in Twitch: Broadcasters and viewers. By studying them separately, we can gain an independent measure of the service's usage levels. We find that Twitch is a global website. Even though 95% of the broadcasters use English, more than 20 other languages have been found: Chinese, German, French and Russian are the next most used languages. Unsurprisingly, for each language, we find that their channels peak in popularity during typical peak times in their respective time zones, e.g., the Chinese channels peak at 20:00 CST.

Figure 4.1 shows how Twitch's popularity has evolved over time(with a 10 day service outage indicated by the gap). It presents the number of viewers and channels (i.e., online broadcasters) over the whole dataset, as well as the monthly average. On average across all samples, there are 9100 broadcasters feeding 362k viewers. An upward trend can be observed, indicating that Twitch's popularity is steadily increasing. We can also inspect August 2014, during which Amazon acquired Twitch, creating significant media

attention. The day before this happened, Twitch saw its peak, achieving 934k viewers. Within context, however, the acquisition has not significantly accelerated growth. Instead, we see a steady growth across the whole period of our data. During the 11 month period, the number of channels being broadcast at any time has grown from 6859 in January to 10k in December 2014. These broadcasts are performed by over 4 million players that we have observed to date. Viewing figures have also been expanding greatly, increasing by 25%.

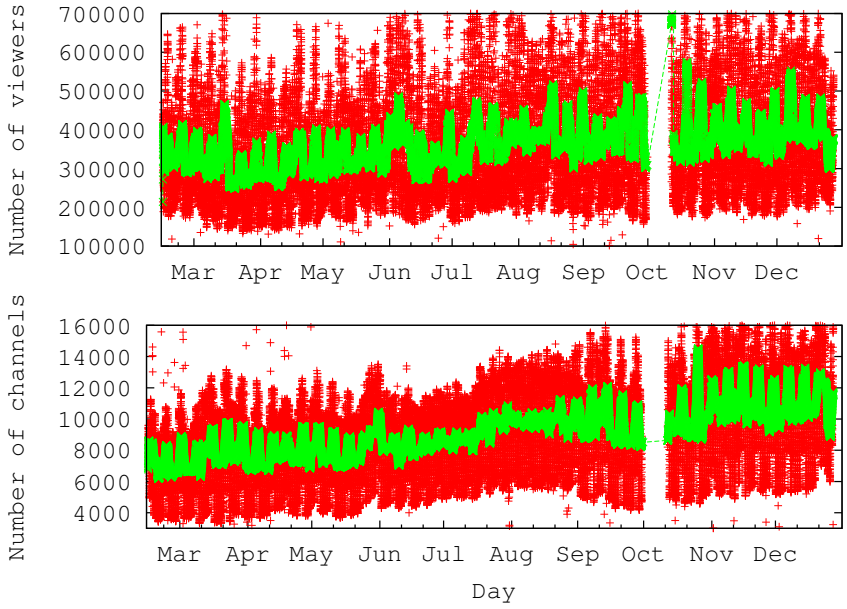

Figure 4.1: Total number of viewers and channels in Twitch over our sample, where red line shows the raw data points we collected (every 15 minutes) and the green is the moving average of 96 raw points (every 24 hours).

4.3 Understanding Content Popularity

The previous section has highlighted the growing usage of Twitch. Next, we explore the nature of popularity, measured by viewing figures, across different games and channels.

4.3.1 How Popular Are Ghose Games

A key question pertains to individual game popularity. Much like music artists vie for air time on radio, games developers may wish to see their games effectively promoted on Twitch. In fact, we see many official promotional channels streaming games pre-release. Twitch can potentially offer an interesting proxy for game popularity, allowing developers, gamers, and reviewers alike the opportunity to get immediate quantitative feedback. Therefore, we begin by inspecting which games are most frequently broadcast and viewed in Twitch.

Figure 4.2 presents the distribution of games by an average number of viewers and an average number of channels from each 15 minutes snapshot. We observe a limited number of extremely popular games, broadcast by many individuals. While 21%(≈4.5k) of the games are played on only **1** channel in Twitch, and 99.36%(≈21k) of the games have less than **10** channels playing. However, the top game "League of Legends" is broadcast by as much as **1255** channels on average, indicating a huge skewness in the distribution of games broadcast.

Figure 4.2 also presents the number of viewers per game, showing the same trend skewed toward a subset of popular games: 92.8%(≈19.6k) of the games have less than 100 viewers, however, the second most popular game Dota2 has 40k viewers on average and the top game "League of Legends" has **105k** viewers on average. It's a massive number considering average Youtube videos viewed per second is 98k [214].

Considering this strong skew, Table 4-B presents the top 10 (0.04% of) games found in Twitch. Collectively, these games garner 64% of all viewers and 34% of all broadcasters in an average of each snapshot. Twitch has, therefore, become an ecosystem largely driven by several extremely popular games. The most notable one is the League of Legends, which throughout nearly the whole dataset has been the most popular. This game, released in 2009, has proven extremely popular with broadcasters and viewers alike. Remarkably, the top three games have remained consistent through the entire

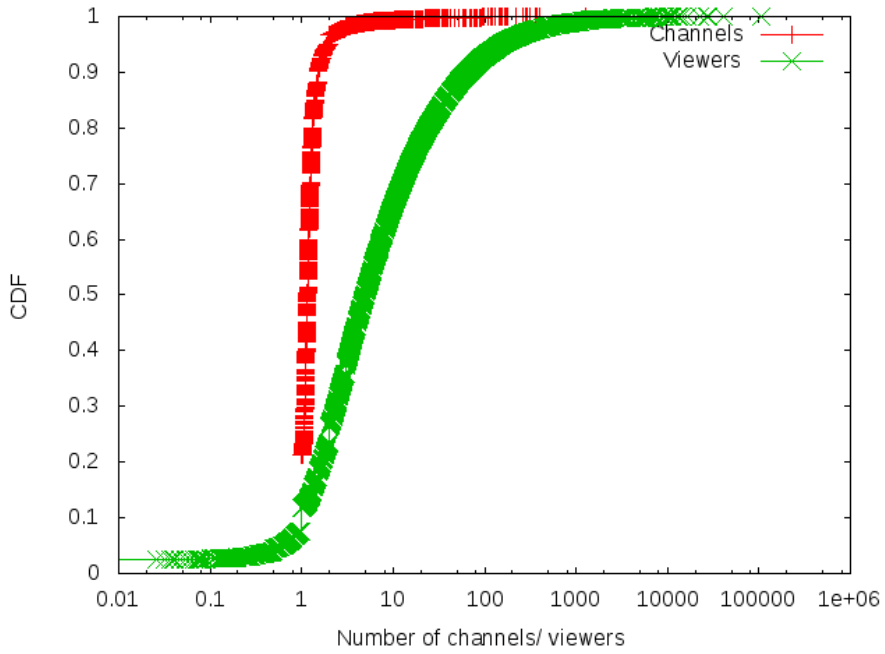

Figure 4.2: CDF of the number of viewers and broadcasters per game

period of our measurements, showing that viewer behaviour is quite entrenched.

Rank	Name	Percentage of Viewer	Percentage of Channels	Percentage of Unique Streamer
1	League of Legends	29.1	14.9	11.9
2	DOTA 2	11	3.2	2.8
3	Hearthstone: HeroesofWarcraft	8	2	3.3
4	CounterStrike: GlobalOffensive	6.2	3.9	5.2
5	Minecraft	3.5	4.2	8.5
6	StarCraftII: HeartoftheSwarm	3.0	1.0	0.6
7	WorldofWarcraft: MistsofPandaria	2.2	2.8	1.9
8	DiabloIII: ReaperofSouls	1.9	1.9	1.7
9	DayZ	1.5	1.5	1.6
10	Destiny	1.4	4.0	6.7

Table 4-B: Top 10 games in Twitch

This highlights a further key difference between game broadcasting and more traditional forms of UGC. Whereas most popular UGC is extremely ephemeral and viral in nature [215], Twitch exhibits much more longitudinal characteristics, whereby games can maintain their popularity for months, and even years. This is *extremely* important

from a game producer's perspective, as gaining prominence on Twitch could potentially bring new customers for the remaining life of the game. This has seen many developers trying to promote their games on Twitch long before they are actually released (c.f., Section 4.3.2.3). We have captured 41 different game streams that are being broadcast as an advertisement on Twitch before their release. Furthermore, Twitch has even made a special video category called "Game Development", which shows programmers developing their games using an online stream. Viewers can even interact with developers before the game has been released: clearly the games industry is taking Twitch seriously: "As we started to build our games, we decided something like Twitch enabled us to expose gamers to the early part of the game development process," said John Smedley, president of Sony Online Entertainment [216].

4.3.2 Categorize the Games

The previous section has highlighted that viewers and broadcasters are *not* evenly spread across games. Rather, a few prominent games dominate Twitch across the whole period of the dataset. Therefore, a key question is what specific attribute made those games popular? To explore this, we next correlate viewing figures with game metadata obtained by the API from Metacritic [217], GamesDB [212] and GiantBomb [213].

4.3.2.1 Game Ratings

A key piece of metadata to explore is the game rating. We explore Metacritic [217], a website that aggregates crowd-sourced reviews to a rating from 1 to 10 (much like video ratings on YouTube). This website, which ranked in the top 1.5k websites worldwide by Alexa [218], influences the game industries, valued by many game development companies [219]. We downloaded all ratings, ranked the games, and then compared them against the Twitch game viewer averaging from each 15-minute snapshot. Interestingly, these values do *not* correlate, suggesting that overall game ratings may not be a strong

predictor for popularity in Twitch. Figure 4.3 shows the average number of viewers of each game by their ratings.

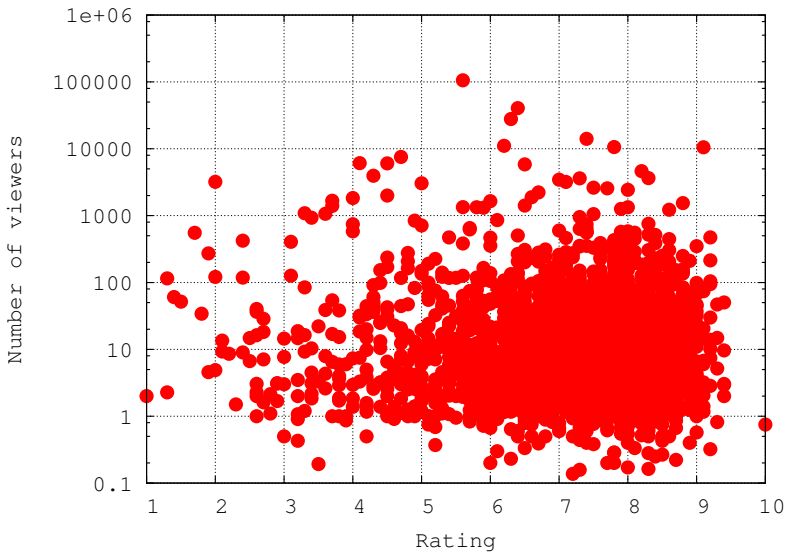

Figure 4.3: Number of viewers of each game, based on their rating

It can be seen that top rated games do not necessarily gain more viewers. Instead, the most popular games viewing figure are rated only 5. This raises questions about the validity of traditional rating services. As Metacritic already been valued by many game developers and companies, the explanation can only be: Twitch follows different popularity patterns to those of Metacritic, i.e., games that can be enjoyed playing versus watching. One interesting case would be a highly rated game named "BEATBLASTERS III" that achieved a score of 8 in Metacritic, but rarely have any viewers in Twitch. We do not have evidence to quantify the veracity of the statement. However, it is clear that Twitch offers significant potential for developers to get quick feedback on their games [220].

Furthermore, when closely looking at the viewer number for games with a rating of 5, we can see the figure is highly biased by the dominant game "League of Legends", far more popular than other games while rating 5. The same for other ratings, each

rating has few dominant games: "CallofDuty:AdvancedWarfare" for rating 4, "Dota2" and "Hearthstone:HeroesofWarcraft" for rating 6 and "Minecraft" for rating 7. Again, we see the popularity is occupied by few extremely popular games.

4.3.2.2 Game Genre

We see from the previous discussion that game rating may not explain the popularity of Twitch platform well. To get a handle on the types of games being played, we categorise them into genres using the GamesDB and GiantBomb service. Figure 4.4 presents Twitch's viewers observed in each genre. It can be seen that a wide diversity of game types are broadcast, ranging from "Action" games to "Massively Multiplayer Online Role-Playing Games (MMORPG)" and "Card" games. As such, there is a large variety of game types for people to select from.

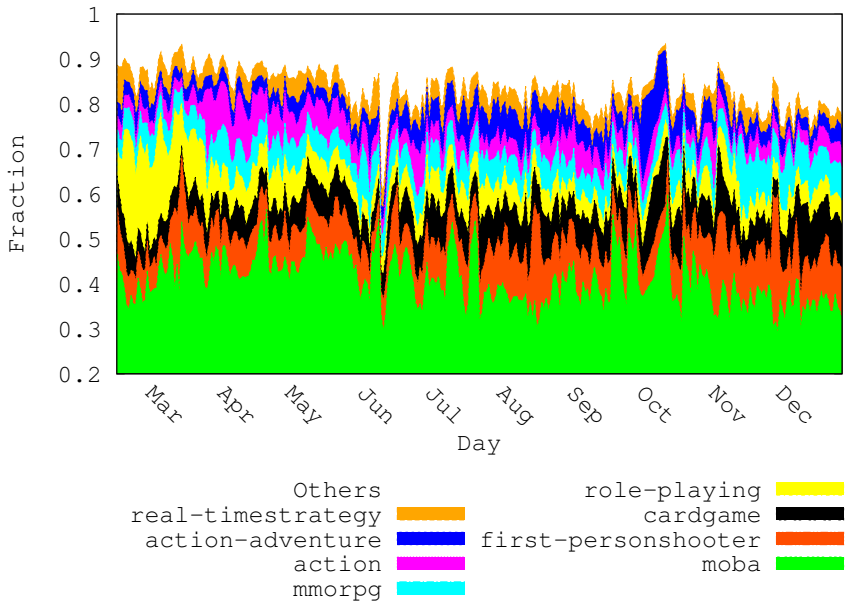

Figure 4.4: Fraction of viewers by genre

Despite the genre "Action" having more games to offer and may take more viewers, it did not stop Multiplayer Online Battle Arena (MOBA) from being the most popular genre. Strategy game "League of Legends" is more popular than shooter game "Coun-

terStrike:GlobalOffensive" or action game "ForbiddenForest". Also temporal variations can be found with new game releases or during events like TwitchPlaysPokemon. This disparity between popularity in genres and number of games leads to potentially a brand new metric about games. As addressed in previous work [116] that online video streaming platforms changed the video game regarding user experience and player interaction. We see an example from Twitch which shows the trend by viewers and streamers. Again, similar to rating, this can be valuable information for game developers and the game industry [220].

4.3.2.3 Game Release Date

After seen the dominant games from rating and genre, the next piece of metadata we collected was the release date. We hold a hypothesis that some new releases may witness a "zeitgeist" phenomenon, whereby they took the viewing figures from old released content. This is an observation common across most content repositories, consumers tend to constantly seek out new stimulation [104, 215, 221].

Figure 4.5 breaks down games into their release years. We first calculate the average number of viewers that each game collected during our dataset and then plot them by their release year. Strangely, we cannot distinguish the game popularity by the release years, as games can receive the same range of popularity across all years. The spark can be found in the year 2009, which is the game "League of Legends".

For the old games, we actually record a notable set of games that were released pre-2000 (26% of all games). Though games from recent years are dominant in Twitch regarding broadcasting channels or time, we still see sparks from games as old as from the 1980s. This is caused by channels which have a primary focus on hosting high-quality speedrun videos and even organise bi-annual charity events. A speedrun is a play-through (or a recording thereof) of a video game performed with the intent of completing it as fast as possible. These play-throughs bring entertainment beyond the

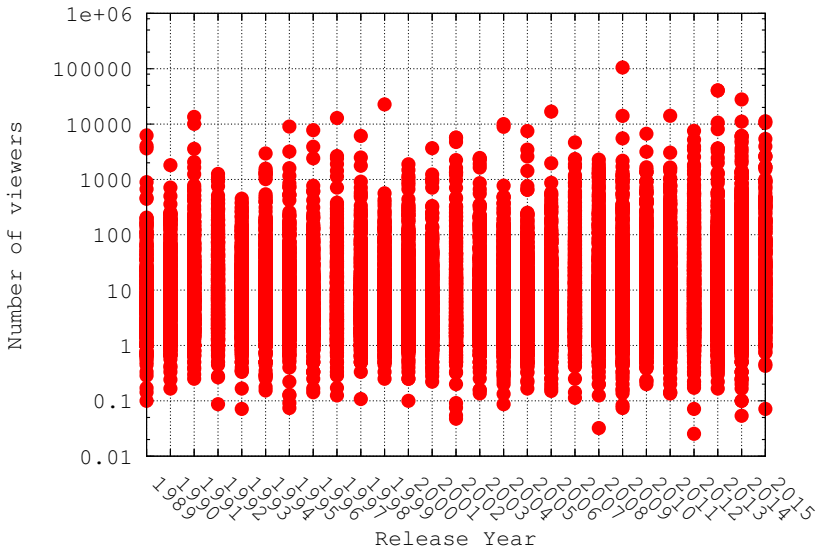

Figure 4.5: Average number of viewers for each game in the system, accumulated based on their release year

game itself, merged with the skill and practice of the player. Famous channels such as "SpeedDemosArchiveSDA" and "GamesDoneQuick" have run over one thousand games, with more than three hundreds of them being spotted in Twitch. Figure 4.6 shows the fraction of viewers each group of games encounters in each day. We can see that old games promoted by channels like speedruns share the similar spikes like new games. This can be characterised as events which we will analyze in the later section.

4.3.3 Summary on Game Popularity

In this section, we have studied the popularity of Twitch against the games. We highlighted a distinct skew towards a few very popular games (Section 4.3). Unlike other video repositories, popularity is relatively static over time with the top three games remaining unchanged across the whole 11 month period. This holds even during periods of new game releases, with fresh games failing to subvert their more established counterparts. The dominant games are not the ones that have the highest ratings on game review-

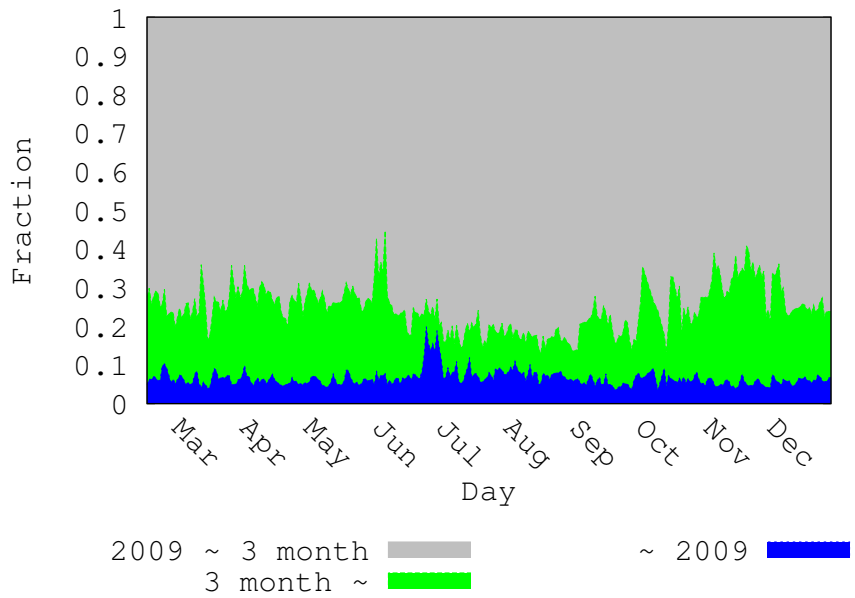

Figure 4.6: Fraction of viewers games corner each day based on the release time of game.

ing website [217] either, showing game streaming popularity cannot be explained by the conventional metrics from the video game industry. Traditional game reviewing websites tend to review *new* releases, but games are updated all the time. This sheds some light on the divergence between game popularity in Twitch and the ratings on reviewers website like GamesDB or Metacritic. Whilst they rate games on how enjoyable they are to play, Twitch ranks games on how entertaining broadcasters are at playing the game. This, amongst other things, can include the style of play, the commentary offered (if any), as well as the social makeup of the users interacting via the chat window.

We found popular games span across in different genres and release years. This is in stark contrast to prior studies of most other user-generated content systems that generally favour fresh content, not that with a far more egalitarian spread of viewers across channels. As we see no definitive domain factors that clearly dictate dominant games in the system. Also by revealing all the top 1 channel in our dataset from each 15 minutes, we found only 20% encountered for events like a new game release. Rather, the "SpeedDemosArchiveSDA" phenomenon gives us a hint that the channel may be

the prominent factor to popularity. We see that the skew towards a small number of broadcasters is also significant with the top 10% acquiring 93% of all viewing figures. Thus, we will start profiling the popularity from the broadcaster angle in the next section.

4.4 Profiling Broadcasters

From the previous section, we found game attributes have little indication of the popularity. In this section, we will look from another angel: channels, to see how number of viewers distributed. We will also investigate each individual channel's behaviour to see how them attracting the viewers.

4.4.1 Channel vs Game

We recorded over 17k unique games across all the channels, suggesting that some games are simply more popular than others. This would potentially bias viewers towards those channels streaming desirable games. To explore this, we group channels as either attractive for the viewer or unattractive. Popular channels are considered to be those that fall into the top k measured by viewing figures, with k increasing from 100 to 10,000 by every 500. This k list contributes from 20% of the total viewing figures when k equals to 100 and reaches 65% to top 10,000 channels. We next compute the games played by these two groups of channels. Surprisingly, we find that each game in the top k attractive channels group is also played by many of those unattractive channels. Playing popular games is *not* the only factor to becoming a prominent broadcaster.

The above indicates that viewers show allegiance to particular broadcasters, rather than games. This is sensible considering the wide variety of broadcasters offering the same game. In such a fluid and competitive environment, it is unsurprising that some broadcasters achieve dominance. Thus, people not only evaluate games based on their independent quality but also on how entertaining they are to watch (and how "well"

they are played by the broadcaster).

4.4.2 Channel Rankings

After knowing the popular channels do not benefit from playing particular games, we are wondering about the boundary between popular and unpopular channels.

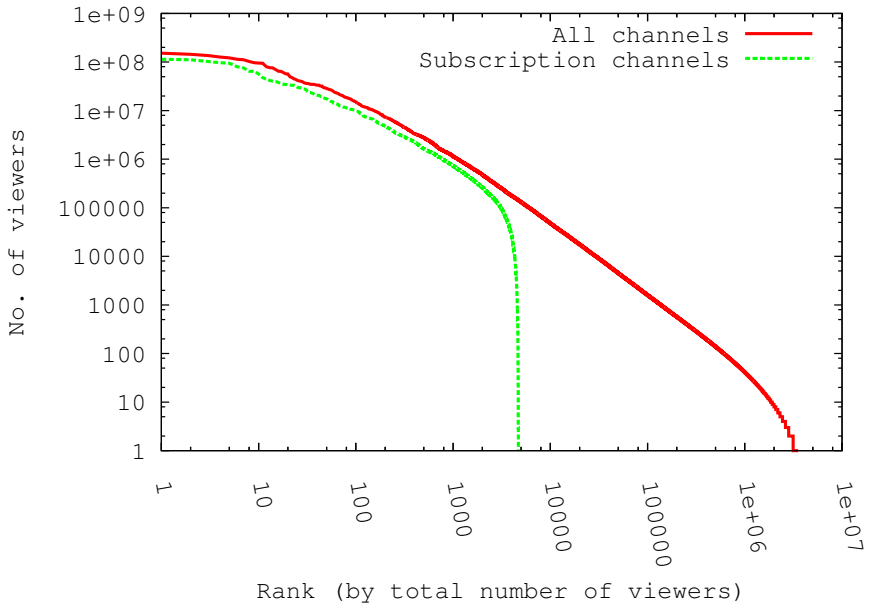

Figure 4.7: Distribution of viewers per game and per channel

Figure 4.7 presents a rank of viewers across the channels by accumulating the number of viewers seen in the dataset for each channel. It can be seen that, much like with games, channel popularity is highly skewed towards a few prominent broadcasters. While 62% of the channels have less than one viewer on average, the top channel "riotgames" cornered 100 million viewers in total and 122k viewers in each snapshot on average. In each 15 minute snapshot, the top 1% channels cornered 70% of the viewers, and the top 10% channels collect 93% of all viewers. As with games, the popularity surely holds by a minority of channels. This minority can also be easily spotted by subscription channels, as the subscription channels are indeed on the popular side of the spectrum. Although we can understand those popular channels are indeed supported by professional streamers,

this level of skew far exceeds that of most other repositories, including UGC [222, 223], adult video [104], VoD systems [94] and catch-up TV [224].

4.4.3 Effect of Games Played per Channel

We further inspect the wider impact of game turnover on channels. Figure 4.8 presents the CDF of the number of games played by each broadcaster. Non-subscribable broadcasters are, generally speaking, not that specialised, with 48% playing more than one game. 62k (1.2%) even exceed 17 games. This confirms that the bulk of streamers play a number of different games. Figure 4.8 also presents the number of games played by subscription-based users. This plot highlights the difference between amateur and subscription-based users clearly, with professionals playing more than 20 games on average.

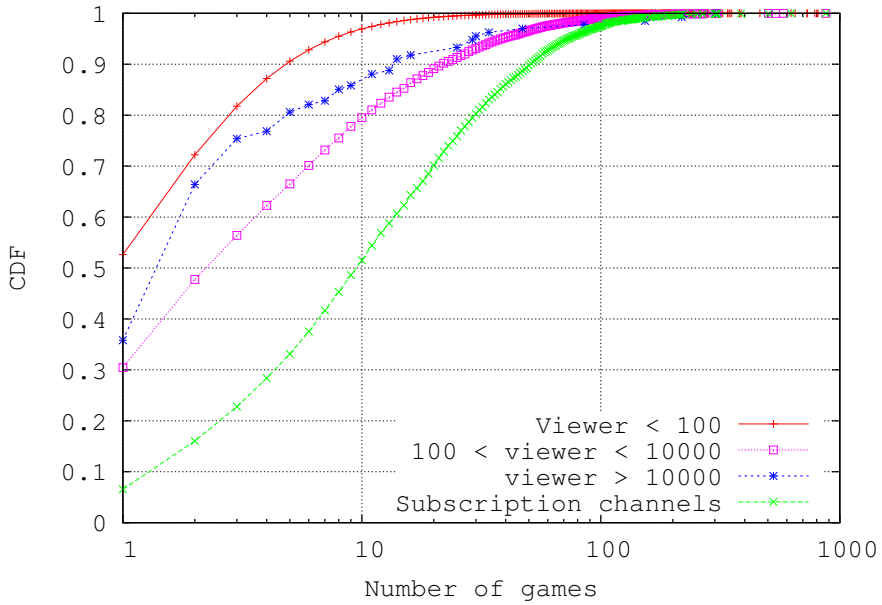

Figure 4.8: CDF of the number of games played per broadcaster

We next explore whether or not the number of games played impacts the popularity of a broadcaster. For example, we are curious what strategy works better: Playing a single game or multiple? Whereas the former would allow a broadcaster to hone

their skills and gain a reputation, the latter may prove more entertaining for viewers. Figure 4.9 plots the average number of viewers a broadcaster receives against the number of games played. On average, broadcasters achieve higher viewing figures when playing more games. That said, it is important not to confuse correlation with causation. We note that many popular broadcasters play several different games, positively biasing the attractiveness of playing many games. A prominent example is user cosmowright, who themes his broadcasts on playing various games in an attempt to complete them in the shortest possible time (i.e., "speedrunning"). By offering this unique experience, cosmowright has achieved peak viewing figures of 11k.

Few can compete with this though: By inspecting Figure 4.9, we see that there are also many experimental broadcasters who play over ten games without gaining high viewing figures. Over 89k users play in excess of 10 games, yet they get, on average, fewer than two viewers. As expected, subscription-based broadcasters consistently attain higher viewing figures, regardless of how many games they play. As such, it is clear that gaining experience and playing different games is not sufficient on its own. Instead, broadcasters must find innovative ways to capture viewers, as with any other entertainment stream.

4.4.4 Game Popularity Affected by Channels

The previous section has explored the number of games broadcasted by streamers, and there is no strong correlation between playing more games and attracting more viewers. We, therefore, begin our characterisation of broadcasters by highlighting some of the more interesting examples. As shown in Figure 4.9, we see some of the channels keep popular after playing many different games. The particular channel we are about to show never broadcasts popular games like 'League of Legends', but, rather, achieves these impressive figures via all very unpopular unknown games such as "Metro".

ManvsGame, is a professional player famed for his unusual style of play. He attempts to play as many diverse games as possible, whilst integrating entertaining live commen-

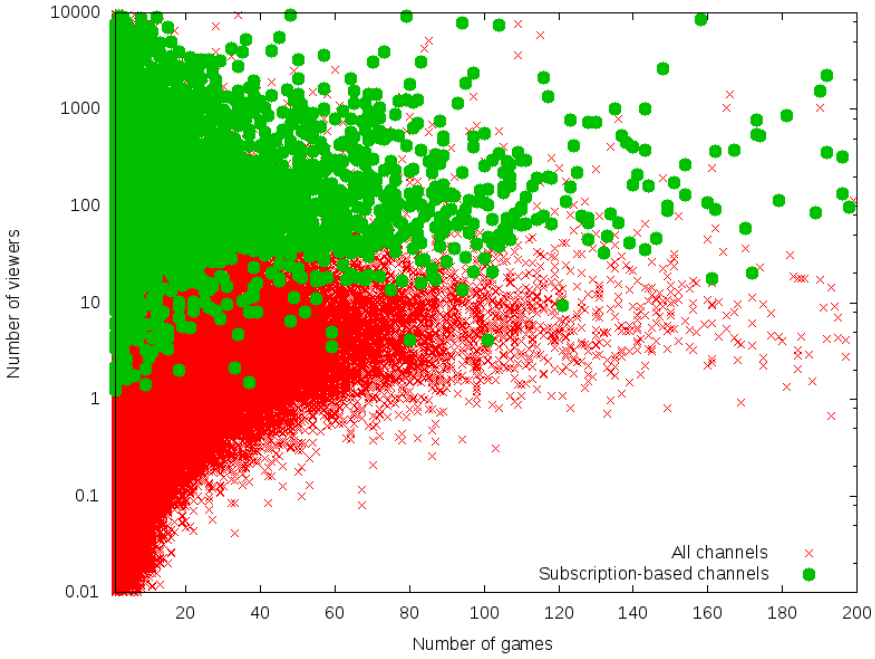

Figure 4.9: The average number of viewers per channel vs. the number of games played by the broadcaster

tary and live interaction with viewers. This behaviour has garnered much attention: We have recorded him playing 42 games, with a peak audience of 24k viewers. This is spread across 180 broadcasting sessions and, on average, we record 4k viewers watching ManvsGame, which is greater than 99% of all other broadcasters.

To explore this, Figure 4.10 plots ManvsGame's activity between 26 February to 10 March.[1] During this short period, ManvsGame plays three games: Strider, Metro and Kingdom. There are others also broadcasting these games simultaneously, however, it can be seen that ManvsGame dominates these games entirely: ManvsGame (red crosses) overlaps most other data points (total viewers in the game) in the plot. When ManvsGame goes offline, the games' viewing figures collapse, reducing from thousands to hundreds (or less). This is a key example of the dominance of broadcasters over games: In many cases, the broadcaster is more of a *Key* factor over the popularity of the game.

ManvsGame is just one example of the unusual types of stream behaviour. Another

[1]These are example dates. This behaviour is consistent across the player's whole lifetime.

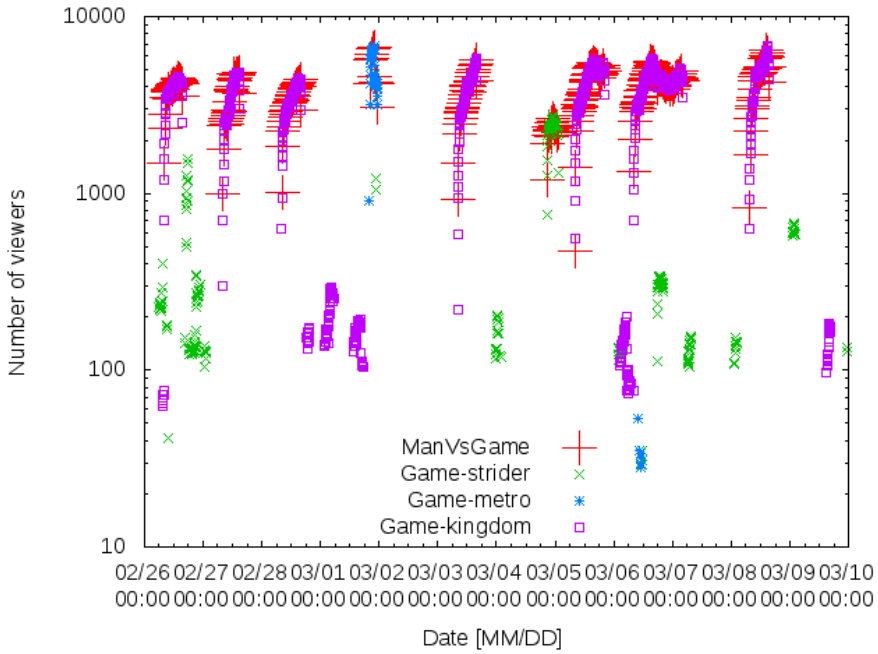

Figure 4.10: Viewing figures over time for ManvsGame, and the viewing figures for all other channels playing the games played by ManvsGame during the same period

interesting example is Twitch Plays Pokemon (TPP). This is a channel that came to prominence in January 2014 and has an unusually long broadcast time, lasting more than 300 days. It operates by receiving votes from the chat feed in terms of what to do. A robot then parses the chat feed and executes the instruction (e.g., move left). This became a viral sensation and quickly gained popularity in Twitch. Work [117] has addressed the uniqueness of this channel apart from showing its effects to the viewers. To understand this, Figure 4.11 presents the viewing figures over time, as well as the viewing figures of the games played by TPP across all channels combined.

During that period, we observed huge viewing figures, peaking at 400k, which constituted 80.5% of all Twitch viewers, nearly the same fraction that a tournament event could take (as shown later in section 4.4.6). This indicates that it is not only large, well-funded tournaments that can gain prominence. TPP was, instead, a single channel that gained prominence without such means at its disposal. Interestingly, we did not observe a notable increase in Twitch's overall viewing figures during this period (see Figure 4.1),

suggesting that TPP viewers were migrating from other channels rather than joining the system anew. Also, we can see that for all the three games broadcast by TPP, the viewing figures of the game are 99% contributed by TPP, including when TPP switches from one game to another. That said, viewers are stick to the channel instead of the games.

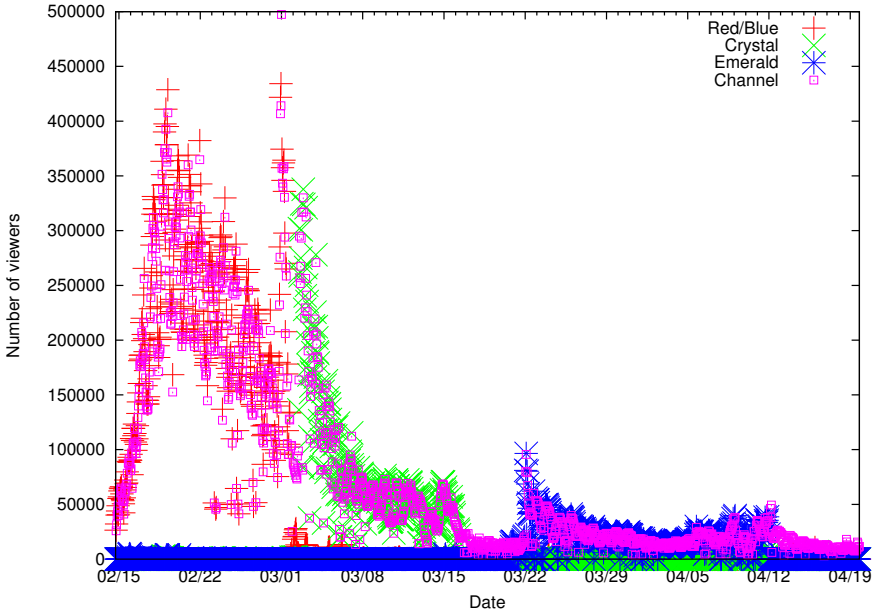

Figure 4.11: Viewing figures over time for Twitch Plays Pokemon and the games played by TPP.

4.4.5 Impact Outside Twitch

The last section discussed the ability of ManvsGame to shift viewing figures by playing many different games in an entertaining fashion. To further explore the influence of the streamer, we are going to reveal if the streamers can shift popularity between platforms. A way of validating this is to confirm the popularity in Twitch with metrics from other social network platforms. We choose YouTube and Twitter as these are two main social network platforms providing video and message sharing services. By associating the user from Twitch with their YouTube and Twitter accounts, we can see how those streamers behave in other platforms. Collectively, we found 78k broadcasters from YouTube and

113k from Twitter.

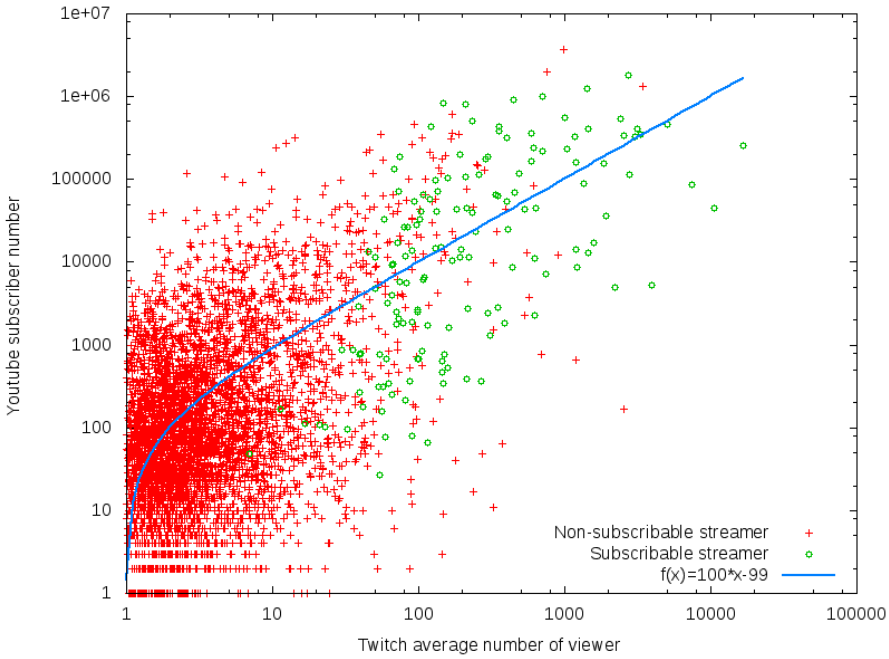

Figure 4.12: Comparing the number of viewers in Twitch and number of subscribers on YouTube.

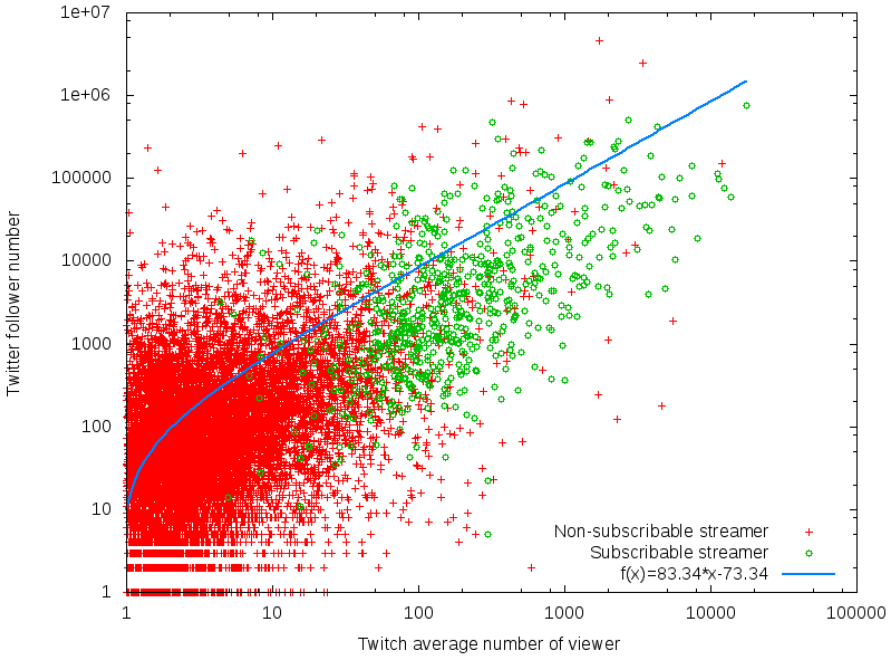

Figure 4.13: Comparing the number of viewers in Twitch and number of followers in Twitter.

As shown in Figure 4.12 and Figure 4.13, the viewer numbers in Twitch are correlated

with the popularity of other platforms, like the subscription number in YouTube and follower number on Twitter. A linear regression line also plotted in Figure 4.12 and Figure 4.13 to help understand the correlation. Especially for those subscribable channels in Twitch, they not only have more viewers in Twitch but are also very popular on other social platforms.

In this section, we started looking at Twitch from the streamer perspective: popularity for channels. The Zipf behaviour of the viewing figure identified that popular channels are extremely popular, holds the key factor to the popularity. We continued investigating how the number of games each channel play could correlate to the popularity. We further showed two individual channels: "ManvsGame" and "Twitch play pokemon", to show that popularity is not really about what to play, it's about how you play instead. And also these two channels show their ability of shift viewers to whatever non-popular games they are playing or took more than 30% of the total viewer in Twitch as a tournament event does. We also found the popular channels are also popular outside Twitch, on YouTube and Twitter. This indicates the popular channels are not only active in Twitch to become popular, they definitely have personal attractiveness more than what we saw in Twitch. We can say that the channels, or the streamers behind them, are more likely to be the dominant element to the popularity of Twitch.

4.4.6 Exploring Tournaments

The above exploration has revealed a particularly important type of channel: tournament broadcasts. The real-time nature of Twitch makes it ideal for broadcasting live events taking place anywhere in the world. This is the case for the burgeoning eSports competitive scene. For example, professional eSports players contend for substantial money prizes (already surpassing $20million [225]). These tournaments gather tens of thousands of spectators, and also achieve huge online viewing figures, with Twitch being one of the primary broadcast platforms [226]. Games like "League of Legends" and "Dota2" are the most common options in those events, and a sizeable prize is usually

involved. In addition to these competitive events, major gaming announcements such as the E3 yearly conference, and other Internet events (*e.g.,* charity marathons) are played through Twitch.

We have manually extracted the key events streamed via Twitch throughout 2014. In total, we have identified 56 events, some lasting multiple days. 53 are eSports tournaments (we left out of this analysis competitive leagues and preliminary phases); the remaining 3 are a charity event, the famous E3 press conference, and the TwitchPlaysPokemon phenomenon [227]. The total number of days from our dataset with an event running are 150 (47% of the considered days).

Figure 4.14 presents each event's share of the overall viewing figures over time (each bar is an event colour coded by the game played[2]). The huge impact of event broadcasters is undeniable; many exceed 20% of the daily viewers, making them the top-ranked channel (Figure 4.7). Whereas the average top-ranked channel's viewing share is 8.9% (daily), this can increase up to 30.5% during events. In absolute terms, we observe close to a million simultaneous viewers in the whole platform. Thus, Twitch resembles other live sports platforms, with spikes during key events [228].

We also inspect the relationship between events and games (see the colour code in Figure 4.14). Tournaments playing League of Legends (LoL) achieve the highest and most frequent peaks. Numerous events playing other less popular games also reach comparable levels of popularity over time. Again, we see that event broadcasters gather the attention of Twitch viewers, even if they are streaming not so popular games.

Figure 4.15 shows a CDF of the share of viewers garnered by the tournaments (on a per game basis); we take samples from every 15 minutes. Overall, we observe that tournaments have a substantial impact on every game (to a lesser extent for DOTA2, which incorporates and incentivises watching tournaments through their own platform). In the most extreme case, for Street Fighter 4, the tournament can reach 100% of all

[2]We treat E3 as a separate game, although this is actually a trade show.

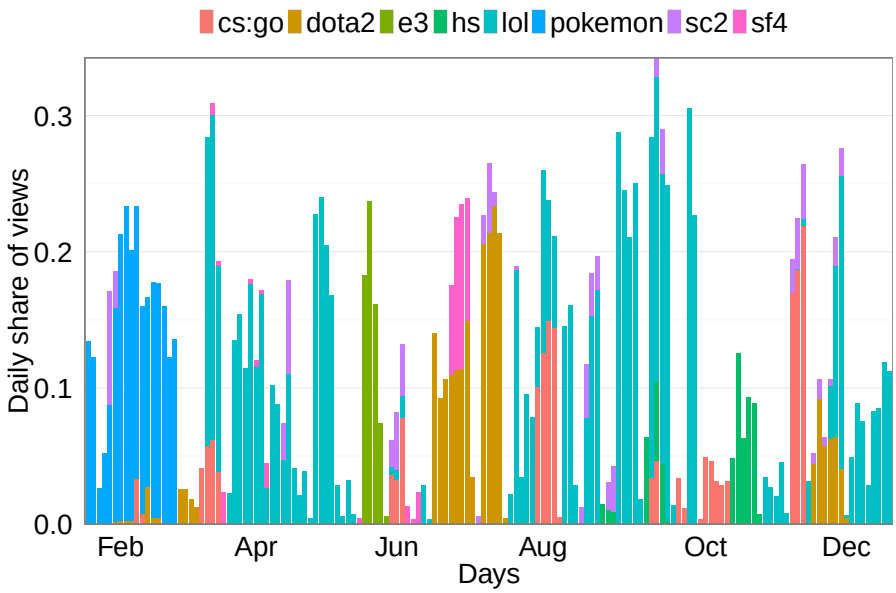

Figure 4.14: Fraction of viewing figures collected by tournaments per day. Tournaments are grouped per game: Counter Strike: Global Offensive, DOTA 2, E3 (gaming trade show), Hearthstone, League of Legends, Pokemon, Starcraft 2, and Street Fighter 4.

viewers of that game. The sometimes large share of viewers captured by tournaments for a given game explains why the companies behind these games provide so much support for the tournaments, e.g., in the form of prize money.

Overall, tournaments nicely illustrate the complexity of the Twitch platform, that lies at the crossing between viewers, broadcasters, as well as different types of companies (e.g., game development and advertising).

4.4.7 Summary on Channel Popularity

The unusual level of skewness towards top games leads us to investigate the channels streaming the games, who broadcast live tournaments, or interactively illustrate the play-through or tirelessly challenge more than 40 different games to the frontier 4.4. Just like games, the popularity of channels is skewed towards a small number of broadcasters, but the behaviour of these popular channels can be very diverse. Closer inspection reveals

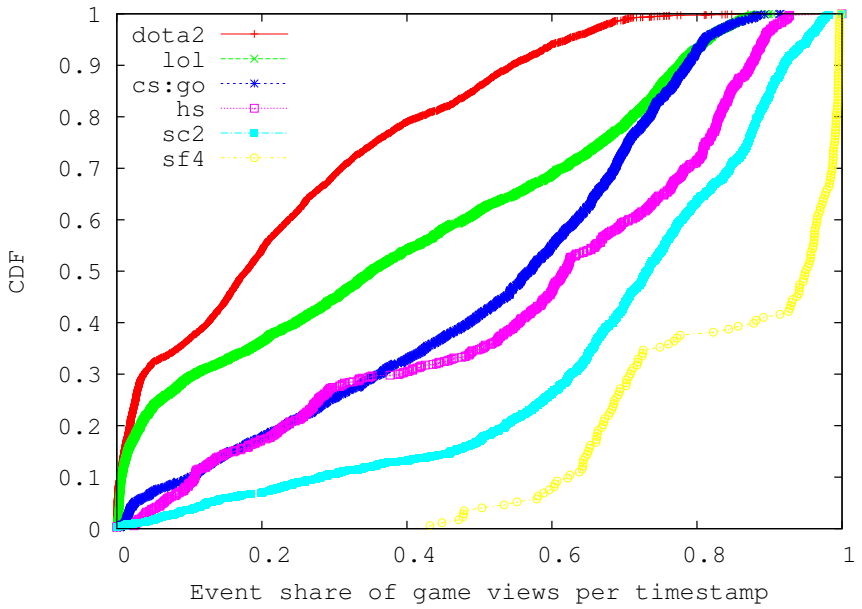

Figure 4.15: CDF of the share of game views gathered by events in each snapshot during our dataset.

that a rising breed of the "professional" player is emerging, with over 5k users offering subscription-based services. These users have quite a distinct effect, broadcasting with their own style, and achieving much higher viewing figures. More evidence of channels dominating popularity are: channels can take all the viewers from the game they are playing; channel popularity is beyond Twitch, spread out other social network systems and organisation channels who broadcast eSport matches can get the viewing number burst during tournaments. We believe that this type of behaviour better relates to sporting events that people are after a few stars or clubs. Similarly, large-scale sporting events such as World Cup bring together a whole community for a short period. Those huge viewing figures will be absorbed by few games which cause the skewness of the game popularity distribution.

4.5 Summary on Twitch Popularity

In this chapter, we have explored Twitch, the most popular game streaming platform in the world. We began by exploring the global usage of Twitch, highlighting its increased popularity over time (in terms of both the number of broadcasters and viewers). We then moved on to inspect the nature of games and channels, focusing on the number of viewers. A surprisingly skewed distribution was found, with the top 10% of games accumulating 80% of all viewers. We also noted that much of this popularity is centred on older games, with newer games failing to gain high viewing figures. We continue looking at a small range of tournaments as those events usually create a huge number of viewing figures. Unlike other user-generated content platforms, these special events seem extremely important and capable of dominating the whole system for a short period and rising the popularity of few related games and channels. ESL (Electronic Sports League, an eSports company which organizes competitions worldwide), at its peak, for example, contained 40% of all viewers in Twitch, push related game and channel to the peak of popularity. This type of behaviour perhaps better relates to large-scale sporting events (e.g., World Cup), bringing together a whole community.

We further focus on characterising broadcasters, as the streamer behind game and event. We found viewers are gathered among a few extremely popular channels, and the Zipf distribution makes other channels very difficult to keep up the viewing figure. By investigating the time and number of games each channel broadcast, we found, especially in subscription-based channels, channels still have a chance to become popular over time. Two individual channels “ManvsGame” and “Twitch plays pokemon” are shown as an example of the influence of streamers. Also, we found the viewing figures of channels in Twitch are highly correlated to their influence in other social platforms like YouTube and Twitter, which further proves the streamer to be a dominating factor towards popularity.

These sport related user behaviours unique Twitch from traditional video platforms who are serving static content, such as YouTube or Netflix. This make us wonder how

would Twitch's infrastructure varies from those traditional video platforms. Especially when serving millions of concurrent users, with the skewness of channels over regions and time can be very challenging to the content management for an Internet-scale application like Twitch. Thus, we intend to further explore the infrastructure of Twitch in the following chapter of this thesis.

Chapter 5

Delivering User-Generated Live Video Streaming at Large-Scale

5.1 Overview

The last chapter has profiled Twitch user behaviour represented by the content popularity. We found a few requirements that distinguish Twitch from the previous video streaming platforms: (*i*) Any user can provide a personal live stream (potentially to millions of viewers); (*ii*) This upload must occur in real-time due to the live social interaction between consumers and producers around the globe; (*iii*) The huge skewness of popularity in different channels and regions requires Twitch manage the content infrastructure more dynamically. Thus, the rapid expansion of user-generated live streaming platforms comes with fundamental challenges for the management of the infrastructure and traffic delivery. For example, the content can not be cached as in other video systems like YouTube or Netflix. Thus, by studying Twitch, a large-scale web service who served more than 2 million concurrent users, we can acquire some important insights into how such challenges can be overcome.

In this chapter, we perform a large-scale measurement study of Twitch. Taking

advantage of a global network of proxy servers, we map the infrastructure used by Twitch. We explore its content replication and server selection strategies, correlating them with both viewer and broadcaster location. Note that broadcaster selection is a unique aspect of personalised video streaming, as prior systems lack the concept of user-generated live broadcasters. In this chapter, we analyse how Twitch has managed to scale-up to deal with its huge demand. In summary, we make the following contributions:

- Section 5.2 illustrates the methodology we utilised to identify the infrastructure footprints and measure it's client redirection strategy.
- Section 5.3 maps the infrastructure and internetworking of Twitch. Unlike YouTube or Netflix which deploy thousands of caches in edge networks, Twitch serves millions of users directly from relatively few server locations in North America (NA), Europe (EU) and Asia (AS).
- Based on this, Section 5.4 exposes how streams are hosted by Twitch at different locations; we explore how Twitch scales-up depending on channel popularity, and how clients are redirected to Twitch servers.
- Section 5.5 further evaluates the client redirection strategy on a global scale. We find multiple factors affecting the redirection policy, including channel popularity and the client network configuration (peering). Due to the lack of peering in Asia, 50% of the clients are exclusively served by NA servers.
- Section 5.6 concludes this chapter.

5.2 Measurement Methodology

We begin by presenting our measurement methodology, which is driven by three goals. First, we wish to discover the location and number of servers of Twitch's infrastructure. Second, we want to know how Twitch manages its resources to stream live channels. Note

that this is a very different model to static video content, which is usually (reactively) cached wherever it is requested. Third, we want to understand how users are mapped to servers through which they can watch the stream they are interested in.

We built a Selenium-based crawler that allowed us to request channels provided by Twitch automatically. The responses to these requests allowed us to inspect which server our client has been redirected to. Periodically, we retrieved the list of active channels using the public Twitch REST API[1], and used our Selenium crawler to extract the set of servers that our client is redirected to. By inspecting the logs, we discovered that Twitch does not manage its delivery infrastructure using CNAME chaining [229], but, instead, manages redirects by giving each content server a unique domain name (and redirecting in the application logic). We observed that Twitch frequently redirects one client to different servers when requesting the same channel multiple times, thus evidencing some mechanism of load balancing.

Previous studies on UGC platforms like YouTube have two approaches to collect the data: either observing the traffic inside ISP [30] or using a distributed system like PlanetLab [127]. However, very rarely people can acquire traffic from ISPs. And the available PlanetLab node is kept decreasing over the years, from 500 in [31] to 398 in [127] (where we found more than 800 vantage points by using the open proxy approach later). Also, very few nodes are available in areas like South Africa, Middle East, Russia and Asia [2]. However, Twitch is considered a very popular streaming platform in these areas, especially in Russia and Asia. Thus vantage points from those areas are really important.

In order to comprehensively sample the infrastructure and explore how different clients were redirected to Twitch servers, we ran the crawler in many geographic locations to achieve global coverage of Twitch's infrastructure. To achieve this, we utilised a global network of open HTTP proxies [3] to launch video requests from around the world.

[1] https://github.com/justintv/Twitch-API
[2] https://www.planet-lab.org/node/1
[3] https://incloak.com/

Figure 5.1: Vantage points adopted: 818 unique IP addresses from 287 ASes, 50 countries in 6 continents.

In total, we routed through 818 proxies, from 287 ASes located in 50 countries as shown in Figure 5.1. We adopted this methodology in two measurement campaigns looking to answer the following questions (*i*) Which locations are used by Twitch to host each stream, and (*ii*) Which clients are redirected to each of these locations?

To sample the geographical footprint of Twitch, we first ran the crawler for several months between December 2015 to April 2016. We launched one request to each channel available on Twitch, through all (818) proxies. We sent a total of 700k unique requests.

Once we had acquired a list of the servers, we began a more focused campaign to understand the redirection strategy employed by Twitch. We explored resource allocation and redirection by simultaneously requesting the same channels from multiple vantage points. We selected eight proxies in multiple locations around the world: two each in US, Europe and Asia, one each in Oceania and South America. We repeatedly requested all live channels through these eight proxies. For each channel, we sent the request multiple times from each vantage point in order to comprehensively sample the servers offered from that location. Each channel was requested a variable number of times (from 15 to 300) based on how many unique servers were found. This provided

a longitudinal view of how content is replicated across Twitch's servers. We collected a total of 321k responses from Twitch.

5.3 Geographic Deployment of Twitch Infrastructure

We start the exploration of Twitch's infrastructure by describing the locations of its servers, as well as how they are connected to the Internet. Our logs show that all Twitch video streams are served from hls.ttvnw.net subdomains. Each domain consists of a server name with an airport code, hinting at a geographical location. For example, video11.fra01.hls.ttvnw.net is a server in Frankfurt (fra), Germany. We confirmed that there is a one-to-one mapping between each domain and an IP address by performing global DNS queries from locations around the world. In total, we discovered 876 servers distributed over 21 airport code subdomains from 12 countries.

It is unclear how accurate these location-embedded domains are and, therefore, we compare the airport codes against the locations returned by three IP geodatabases: ipinfo.io, DP-IP and Maxmind GeoLiteCity. Although the airport locations embedded within the domains are always in the same continent, we note that they are inconsistent with the locations returned from the databases. Instead, the geodatabases report that Twitch operates a centralised infrastructure. All servers were mapped to just four countries: Switzerland (Europe), Hong Kong (Asia), US (North America) and Sydney (Oceania). In total, our traces reveal 360 servers in the North America (NA), 257 servers in Europe (EU), 119 in Asia (AS) and 47 in Oceania (OC).

To explore the discrepancy between the databases and airport codes, we performed a TCP-based traceroute and ping campaign from 10 sites in East and West US, Europe, Asia Pacific and South America. From the traceroute path we see that servers are sharing a prefix also pass through the same router when entering Twitch's AS, with only the last three hops differing. This, however, does not confirm physical locations. Hence, we also check the Round Trip Time (RTT) to each server using TCP ping. We

plotted the RTT value found from different vantage points on Figure. 5.2 against each of the Twitch servers labelled by the IP address. This figure shows a clear boundary between servers with different airport codes. Servers inside the same sub-domains tend to differ by under 5ms; for servers on the same continent, the difference is within 50ms; for servers on different continents, this increases beyond 100ms. We found a minimal RTT of under 3ms when accessing servers sharing the same country code. This suggests that the airport country codes are a good indicator of physical location. In other words, this highlights inaccuracy in the geolocation databases (this is perhaps reasonable, as geodatabases are well known to suffer limitations such as address registration [230]).

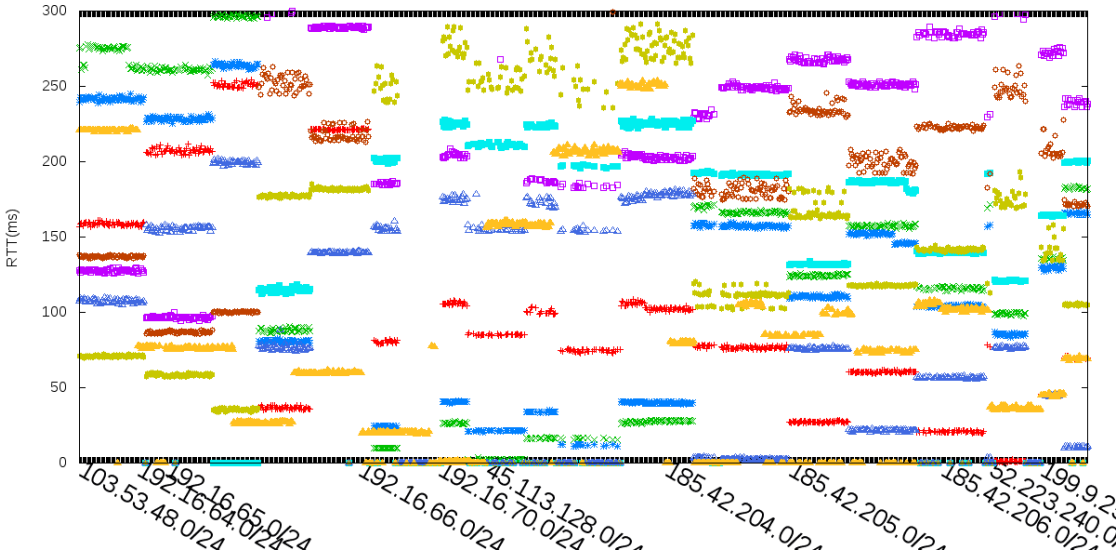

Figure 5.2: The RTT value found on each Twitch server, with vantages points distinguished by color. We can see a clear boundary between physical clusters.

We gain additional confidence in our findings by checking the BGP routing tables.[4] Unlike other large content providers, we fail to find any third party hosting, as seen in other larger CDNs like Google [230] or Netflix [34]. Instead, all servers are located within Twitch's own Autonomous System (AS46489). Importantly, we find the prefixes are only announced in their appropriate continents. For example, 185.42.204.0/22 is only announced in Europe and 45.113.128.0/22 is only announced in Asia. Thus, we are

[4] http://routeserver.org/

confident that the geolocations are at least accurate on a continent-level granularity

Finally, to dig deeper into the BGP interconnectivity of Twitch's AS, we utilise PeeringDB [231] to extract the locations of advertised public and private peering facilities used by the 153 Twitch peers listed in [232]. Figure 5.3 presents the number of *potential* peers that are collocated with Twitch in Internet Exchange Points (IXPs) and private peering facilities. Unsurprisingly, we find a tendency for more peering in countries where we also discover Twitch servers. For example, most of the *potential* peerings are located in IXPs in the Netherlands (AMS-IX), US (Equinix), UK (LONAP) and Germany (DE-CIX Frankfurt). Noteworthy is that the number of *potential* peerings in Asia is actually quite small, with the bulk in America and Europe (we acknowledge this could be caused by inaccuracies in PeeringDB). We find from BGP route records[5] that the IP prefix for the Asia presence was first advertised in June 2015. This recency could explain the low number of peers. The same is for Oceania, which first was advertised in November 2015. The low number of peers could affect the performance in redirection, as we will illustrate later in Section 5.5.

The above results only allow us to definitively state that geolocations are accurate on a per-continent basis. Hence, for the rest of this chapter, we focus our analysis on *continent-level* geolocation; where countries are mentioned, we use airport codes as the ground truth. Due to the low utilisation of Oceania servers, we will mainly focus on NA, EU and AS in the following sections.

5.4 Stream Hosting Strategy

The previous section has explored the location of Twitch's infrastructure. However, this says little about how it is used to serve its dynamic workload. Next, we look at how streams are allocated to Twitch's servers.

[5]https://stat.ripe.net/

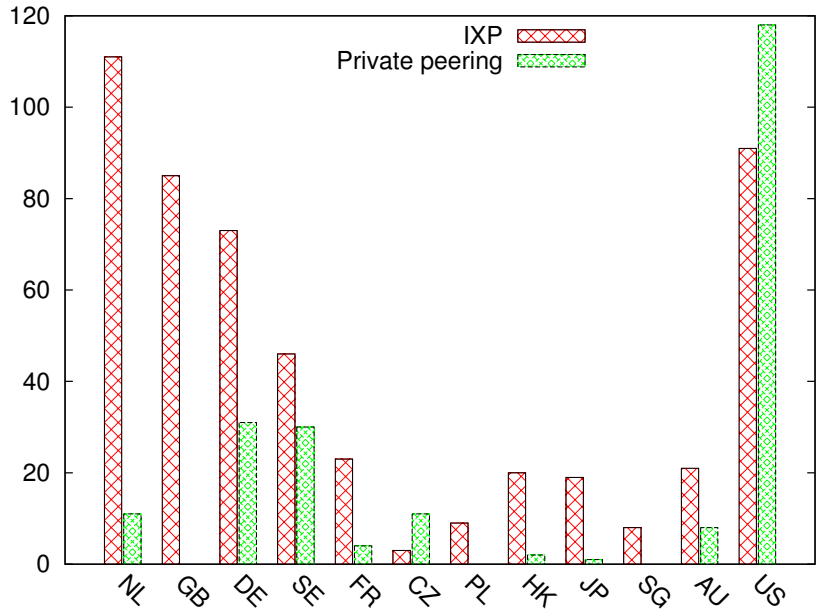

Figure 5.3: Number of peers collocated with Twitch AS46489 at Internet Exchange Points and private peering facilities in each country (from PeeringDB). There is more peering in countries where Twitch servers are based.

5.4.1 How Important is Channel Popularity?

We first look at the number of servers a channel is hosted on, based on how many viewers it receives (*i.e.,* popularity). It might be expected that the number of servers hosting a channel scales linearly with the number of viewers. However, we find this is not the case for Twitch. Figure 5.4 presents the number of servers hosting a channel against the instant number of viewers per channel. Live viewer figures are acquired from the Twitch API. Although there is an upward trend, it is not that distinct (highest correlation is just 0.41). We also explored the total number of viewers (accumulated viewers over time). However, the correlation with a number of servers was not higher.

The low correlation suggests a more sophisticated methodology is used to manage the scaling — it is not solely based on the number of viewers. To understand this better, we take a temporal perspective to see how the number of servers utilised for a channel evolves over time. We manually selected 30 popular streamers from different countries

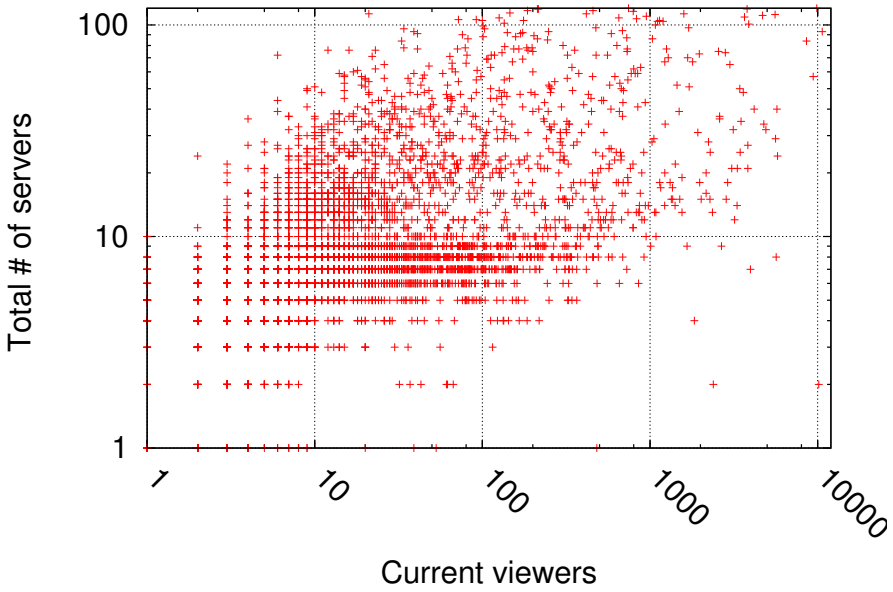

Figure 5.4: Number of unique servers hosting each channel (found using requests from multiple vantage points all over the world) against number of current viewers. Channels with high view counts are replicated on a larger number of servers.

and repeatedly requested their channels every 5 minutes from the proxies.

Figure 5.5 presents example results from a US streamer and a Chinese streamer. Both channels have an initial allocation of 3 servers when they start the streaming session. As more viewers join, the popularity is followed by an increase in the number of servers provisioned by Twitch. The figure also shows how drops in viewing figures are accompanied by a decrease in the number of servers. When looking at the number of servers per continent, it can be seen that the capacity is adjusted independently per region, with the Chinese streamer having only three instances in Europe and America. Again, this confirms that Twitch scales dynamically the number of servers allocated to a channel, depending on the view count. Moreover, it indicates that each region is scaled independently based on the number of viewers in that region. That said, despite the content is live, more servers are required to offload the requests in the region. But how servers are assigned inter-regions?

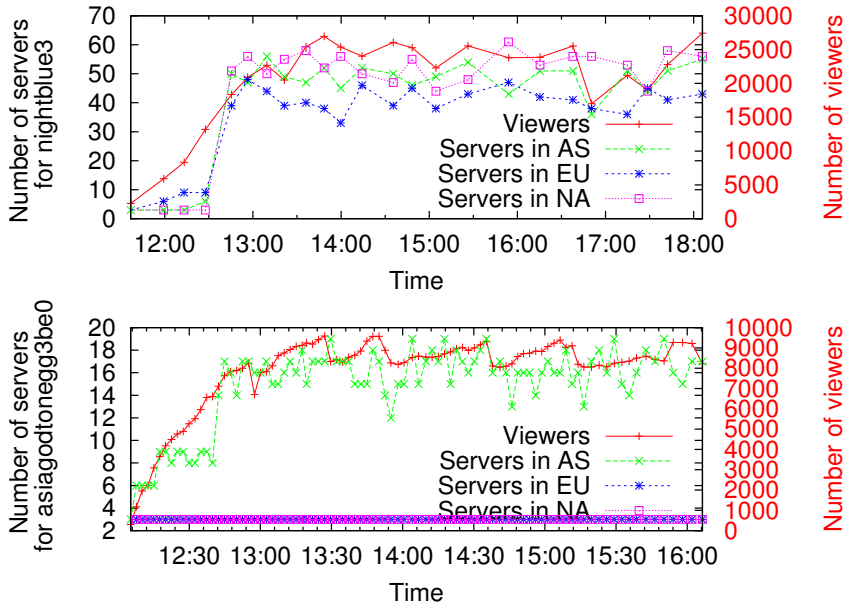

Figure 5.5: (*a*) Number of servers found for channel nightblue3 (US streamer) as a timeseries; (*b*) Number of servers found for channel asiagodtonegg3be0 (Asian streamer) as a timeseries. The number of servers are scaled independently in each region.

5.4.2 Scaling of Servers Across Continents

The previous section shows that the number of servers hosting the channel is correlated with the number of viewers watching the channel per region. We next investigate how the scaling works across continents. Figure 5.6 presents the fraction of servers found in each continent for each channel (based on its number of viewers). We present both the bottom 70% and top 10% of all channels during one snapshot.

We can see from Figure 5.6 that channels with a small number of viewers tend to be predominantly served from NA only (red). 67% of channels with 0 viewers are exclusively hosted in the US; this drops to 63% for one viewer, 48% for two viewers, 40% for four viewers, and just 24% for five viewers. As the number of viewers increases, the fraction of US servers hosting the stream decreases (to be replaced by both EU and AS servers). Channels with over 50 viewers are nearly always served from all three continents. Figure 5.6 also shows the server distribution of the top 10% channels, with

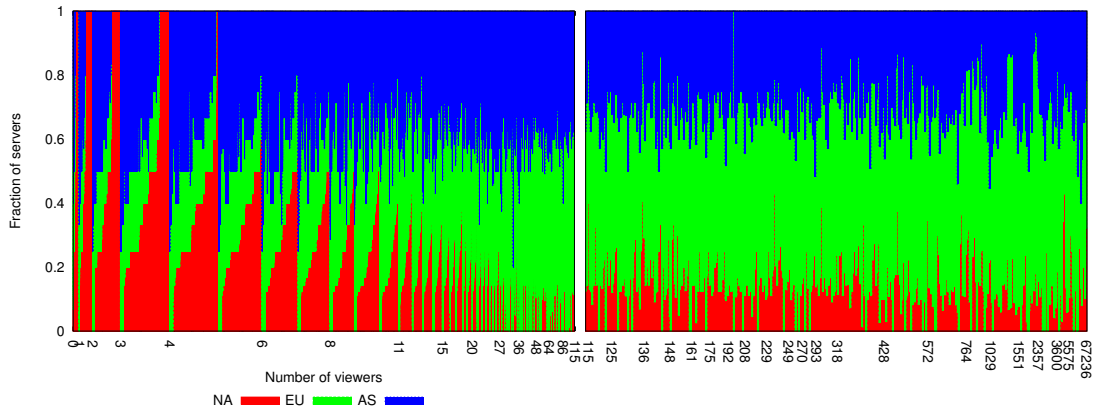

Figure 5.6: Fraction of servers found from NA, EU and AS cluster for the bottom 70% (left) and top 10% channels (right). Only popular channels are replicated outside of NA

21% of servers in NA, 53% in EU and 26% in AS overall.

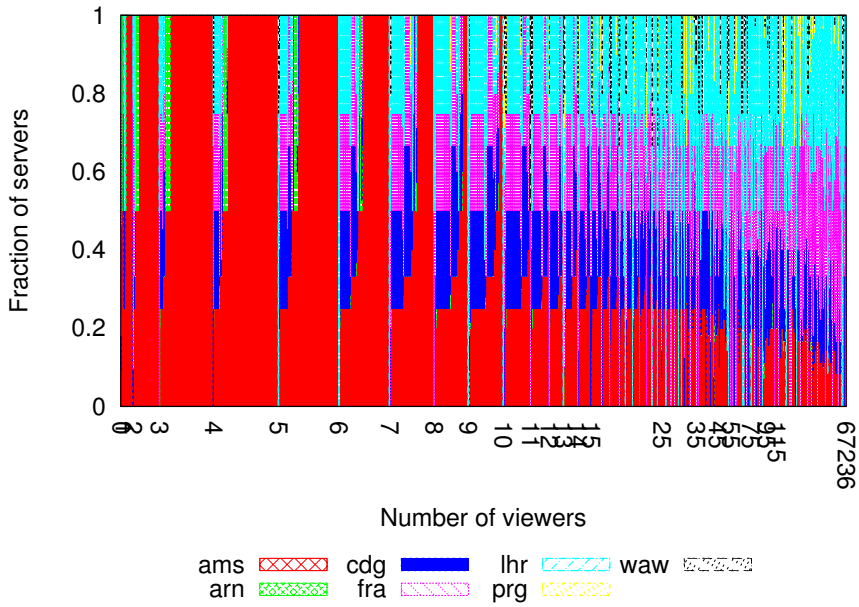

Figure 5.7: Fraction of servers from EU clusters including Amsterdam (ams), Sweden (arn), Paris (cdg), Frankfurt (fra), London (lhr), Prague (prg) and Warsaw (waw).

We can also explore the servers that channels get replicated onto on an intra-continental basis. We find that, for the EU cluster, servers from Amsterdam (ams) are primarily used, especially when channel viewers are less than 10. Along with several remaining domains, *i.e.,* Paris (cdg), Frankfurt (fra) and London (lhr), servers are shared equally in 4 sub-

domains. We find other very underutilised locations, for example, 34 hosts are found in Amsterdam, while Warsaw (waw) has 38. However, we rarely see any servers in Warsaw. Not surprisingly, the fraction of servers found in each is correlated with the number of peerings as shown in Section. 5.3. This can be further confirmed by the AS cluster that the servers are often equally distributed between two or three POPs from either Seoul (sel), Tokyo (tyo) or HongKong (hkg). That said, the server redirection that allocates streams to servers based on how the user entered Twitch, plus network peerings lead to allocating servers skewed towards heavy peering locations.

Briefly, we also see distinct patterns within each continent. For example, in NA, channels are always first hosted in San Francisco (sfo) before being scaled out to other server locations in the region. The same occurs in EU and AS, with Amsterdam (ams) and Seoul (sel) usually hosting a stream before other continental locations.

5.4.3 Summary of Hosting Strategy

We can combine the above, to highlight the scale-out process in more detail. We find that *all* channels initiate their stream on a single server in the NA; as the channel increases in popularity, Twitch first replicates a copy on each continent, before slowly increasing the number of replicas in the locations that receive most requests.

This suggests that the NA operates as a head, which pushes content out to subordinate clusters in Europe and Asia. Following this, each cluster is responsible for selecting how many internal servers it allocates based on the stream's local popularity. The allocation of channels onto servers is only one part of Twitch's strategy. In the next section, we will specifically profile the redirection strategy from different continents.

5.5 Client Redirection and Traffic Localisation

The previous section has shown that Twitch tries to adapt to the global demand by progressively pushing streams to multiple servers on multiple continents. In this section, we explore the mapping of clients to these regions by utilising our full set of proxies. We perform a full channel crawl from each location and see where the clients are redirected to (*cf.* Section 5.2). Table 5-A provides a breakdown of the redirections between different continents. In the majority of cases, Twitch assigns a server from the nearest continent: 99.4% of the requests in North America and 96% of requests in South America are handled by servers in NA; 82% of the requests in Europe and 78.2% of the requests in Africa are served by EU servers.

Table 5-A: Traffic distribution of Twitch clusters globally.

Fraction(%)	**NA cluster**	**EU cluster**	**AS cluster**
North America	99.4	0.6	0
South America	96	4	0.01
Europe	17	82	1
Africa	21.8	78.2	0
Asia	34.4	20	45.6

Our results also contain some noticeable outliers. Asian servers handle only 45.6% of requests from Asian clients; more than one third of the requests are handled by NA servers. That said, the NA cluster also absorbs the vast majority of requests from other regions that are not resolved to their local servers, including AS and EU. In order to explore the reasons behind this apparent mismatch, we investigate for each proxy the fraction of redirections to its local (continental) servers when requesting the full list of channels. Figure 5.8 shows the empirical CDF of the fraction of local servers observed by each proxy. We separate the plots into each continent for comparison. A clear contrast can be seen in the three different regions: nearly 90% of the clients in North America are always served by NA servers, and almost 40% of the clients in Europe are always served by EU servers. However, for Asia, 50% of the clients are never served by the Asian servers, and only 10% are entirely served by Asian servers.

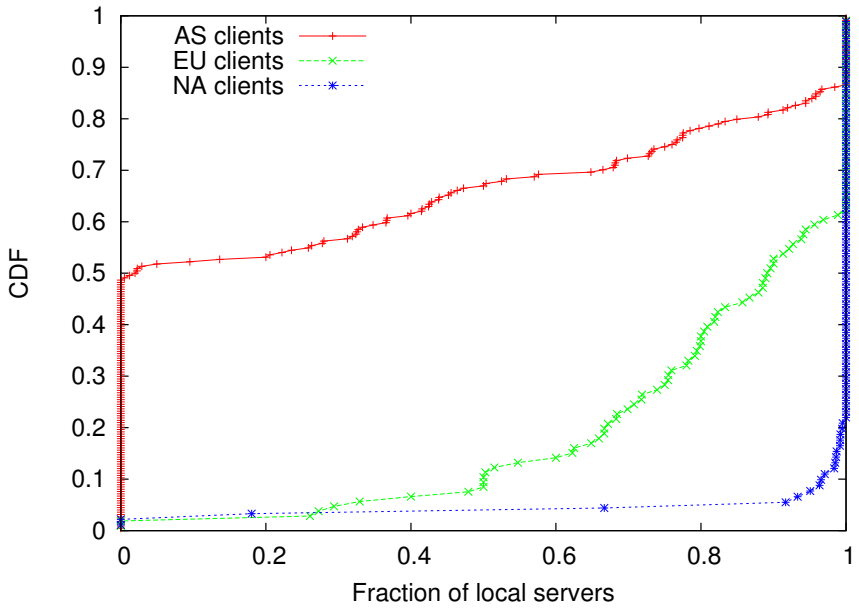

Figure 5.8: Fraction of local servers observed for each proxy. Clients are grouped by continents for comparison. NA users are usually served locally, whereas most AS clients must contact servers outside of AS.

As previously noted, the number of servers that host a stream is closely related to the stream's popularity. Hence, we also inspect the relationship between channel popularity and the ability of clients to access streams from their local cluster. Figure 5.9 presents the fraction of requests that are redirected to a cluster on the same continent, plotted against the popularity of the channels. Again, it can be seen that European clients get far more local redirects, whilst Asian requests regularly leave the continent. This is consistent across all channel popularities, although in both cases, more popular channels receive a large number of local redirects.

An obvious question is why do the Asian clients suffer from such poorly localised redirects. Only 15% of our Asian clients exclusively utilise Asian servers; 50% are *never* redirected within Asia. To analyse why this might be the case, we revisit the peering policies of those particular networks. When inspecting the 15% of Asian clients that exclusively rely on Asian servers, we see that they all share the same private peering facilities with Twitch (based on PeeringDB). For example, AS36351, AS9381 and Twitch

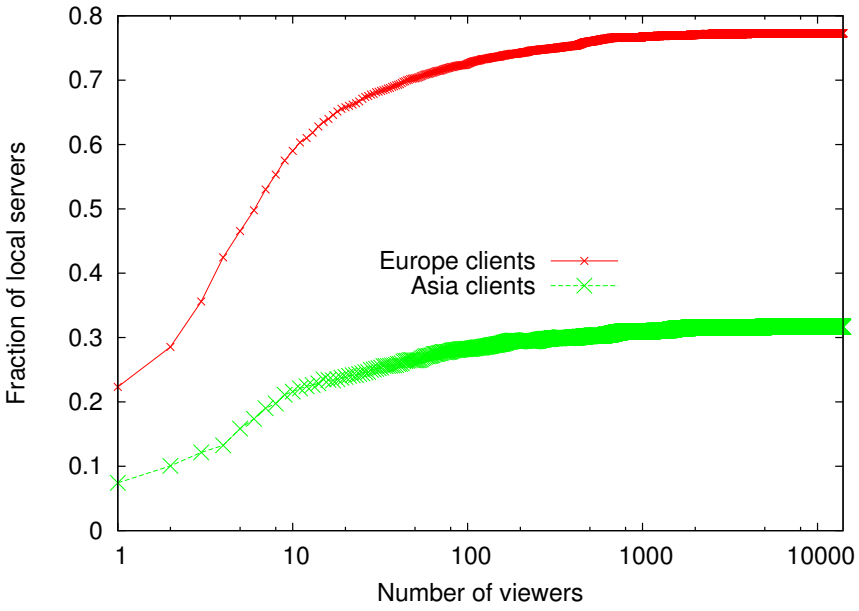

Figure 5.9: The fraction of local servers used *vs.* the total number of viewers for a channel. More popular channels are more likely to be locally available on a continent.

are all registered in Equinix, Hong Kong. In contrast, the remaining networks do not peer. Therefore, it is likely that Asia fails to localise its requests because of these poor existing peering arrangements (Section 5.3). Even if the servers in Asia are geographically nearby, their network distance might be higher. Similar scenarios can be found in previous work [233], highlighting that topology and peering are far more important than geographic distance.

5.6 Summary on Twitch Infrastructure

In this chapter, we have studied the Twitch streaming infrastructure. Through empirical measurements, we have shown that Twitch's architecture is surprisingly centralised, with just three server clusters. This is because the benefits of using decentralised caches (in edge networks) are decreased radically in live streaming (as caching cannot take place). We have also shed light into how channels are scaled-up across clusters and shared amongst clients. We found that the channel popularity is a strong factor in content

placement and local server selection. Twitch has clearly built an intelligent scale-up (and scale-down) strategy, whereby popular streams are progressively replicated across more servers to handle demand. We also find that peering strategies in Asia reduce the impact of a local cluster, instead, resulting in many redirects to the US.

By studying Twitch, we reveal the infrastructure and hosting strategy of the largest user-generated live video streaming platform. Indeed, to our surprises, Twitch uses a more straightforward deployment than VOD, UGC and live video streaming: less than 5 clusters within the same AS around the globe. However, this also makes sense due to the live attributes of the streams so that content can reach the viewers as soon as possible. We also spotted that peering can affect the redirection performance directly which, addresses the importance of network peering on the Internet. Also, our findings are confirmed by contact with engineers from Twitch. One year after our original measurement results published, Twitch confirmed that this CDN deployment has not been fundamentally changed, and they have extended their footprint to some other developing countries such as South Africa.

Combining with the user behaviour study, we have acquired a full picture of large-scale distributed systems from the content popularity to the live stream delivery. We found, indeed, the user-generated live video streaming platform Twitch has it's own character in user behaviour compare to traditional VOD platforms, thus the infrastructure is also built to coordinate with the character exposed by the live content served. That said, we consider our research goals on the large-scale distributed applications fulfilled and we will start looking at data-center scale distributed applications in the next part of this thesis.

Part IV

Profiling a Data Center Scale Distributed Processing Application

Chapter 6

Profiling Data Centre Scale Distributed System Traffic

6.1 Overview

Hadoop and related distributed data processing applications have become the standard for large-scale data computation. As an essential component to build up the cluster, the network plays a very important role to exchange the data and processing tasks among different nodes. Benchmarking studies show that communications constitute a significant part of the overall execution time [142]. More than passively transport the traffic, the network can also actively acceurate the data processing by performing in-network aggregation operations [144]. However, the network behaviour of Hadoop is not well-understood yet, thus it's difficult to evaluate the network performance of Hadoop, not to say perform sophisticated in-network processing operations to accelerate the distributed processing.

In this chapter, we aim to characterise the network traffic behaviour of Hadoop. To capture the original behaviour of Hadoop, we first setup a physical cluster and collect all the Hadoop traffic while running a set of benchmarking jobs. By varying the cluster

setting, jobs and job settings, we not only identify the Hadoop traffic driven by behaviour indicated by the source code, but also investigate how the Hadoop traffic is correlated among different setups. Based on the characterisation, we propose a 5-tuple traffic model that describes Hadoop traffic with MapReduce application specific behaviours taken into consideration. Overall, the traffic characterisation provides fundamental knowledge on delivering Hadoop traffic. By using the traffic model, we can quantify the Hadoop traffic, for example comparing the traffic from different jobs. Also, reproducing the traffic following from the model is made possible. This chapter is organised as follows:

- Section 6.2 shows the methodology we use to profile the Hadoop traffic, including the cluster and experiment setup.
- Section 6.3 characterises the Hadoop traffic under various jobs, job settings and cluster settings.
- Section 6.4 proposes a 15 elements traffic parameter that models the Hadoop traffic. The fitting result of using such traffic parameter is shown, also under various cluster settings.
- Section 6.5 concludes this chapter, highlighting the remaining work for the next chapter that evaluating the network performances with Hadoop traffic.

6.2 Measurement Methodology

6.2.1 Cluster Setups

As mentioned in the background chapter 3.1.2, Hadoop generates several types of traffic, including HDFS control messages, HDFS data transmissions, Shuffle data transfers, and Yarn control messages (*e.g.,* keep alive). Hence, it is necessary to understand the importance of these several traffic components. We setup a Hadoop cluster with 16 physical work nodes and a master node. We first use the default cluster settings (as listed in

Table 6-A: Default physical/test environment settings.

Name	Value
data replications	3
cluster size	16
block size	128MB
input split	128MB
number of reducers	2
output replication	1

Tab 6-A) to observe common traffic behaviour, and then change the settings to show the traffic variation.

To capture a range of traffic, we sample a diverse of Hadoop jobs, including machine learning, graph algorithms, and scientific computation. We focus on a few types of jobs than commonly used in the Hadoop studies [144, 234], taken from the benchmarking tool HiBench [148]. The jobs are (*i*) *TeraSort*: the TPCx benchmarking job for Hadoop clusters [146]; (*ii*) *PageRank*: a graph processing operation for calculating the weight between vertices [149]; (*iii*) *kmeans*: one of the most popular data clustering algorithms [235] and (*iv*) *Hive Join/Aggregation*: two of the most commonly use functions in SQL, however build upon Hadoop with Hive. We considered other types of jobs in our initial experiments (*e.g.,* word count, join queries, Bayes), and found that the selected three provide a good sample of different types of MapReduce computation as they contain most traffic patterns.

Our Hadoop cluster is built using Cloudera Distributed Hadoop(CDH)[1] version 5.4.7. The cluster contains 15 work nodes and one master node. Each node contains eight cores, 32GB memory and a SSD 400GB disk. All nodes are connected to the same switch with 10 Gb links. Collectively, this cluster can run jobs requiring up to 500GB of memory and 6TB of storage. This is considered sufficient to run those jobs we selected. Although this topology is considered straightforward compare to other modern topologies, however, we aim to capture the traffic demand from the application level rather than the network level, thus this straightforward topology help us enough on expose the application traffic.

[1] http://www.cloudera.com/

A sflow agent [2] is installed on each machine with a dedicated sflow collector on the same switch. Then we regularly collect processes and flow information from the operating system on each machine, and cross reference them with the sflow collected. In the end, we can map the traffic to the origin process by linking the process and network information collected on the node to the traffic information collected via slfow. Thus, we have a full traffic map contains the process info, the source node and port, the destination node and port and finally, flow size and duration.

6.2.2 Experimental Setup

To obtain an accurate coverage of the network behaviour, it is necessary to run jobs multiple times. Indeed, data placement, for example, is not deterministic in Hadoop [3], and therefore multiple runs are necessary to sample different data placements. To get an idea of the number of runs required, we launch 1000 identical TeraSort jobs. Each time we clean the cache and calculate the aggregated traffic volume generated. We then use the ks.test to test if the traffic volume distribution after a certain number of runs is the same as the one found with 1000 samples. Figure 6.1 presents the p-value evaluated after each run. There is significant variance when performing less than 200 runs. However, we find that the p-value is close to 1 (with a very small distance D) after 400 runs, indicating that the same distribution is found. The same convergence in distribution happens not only in the case of the traffic volume, but also for other metrics, such as the number of sessions, and the number of nodes involved. Therefore, all our subsequent traffic analysis is based on 400 runs for each individual setup.

[2] http://www.inmon.com/technology/sflowVersion5.php

[3] See chooseRandom in org.apache.hadoop.net.NetworkTopology.

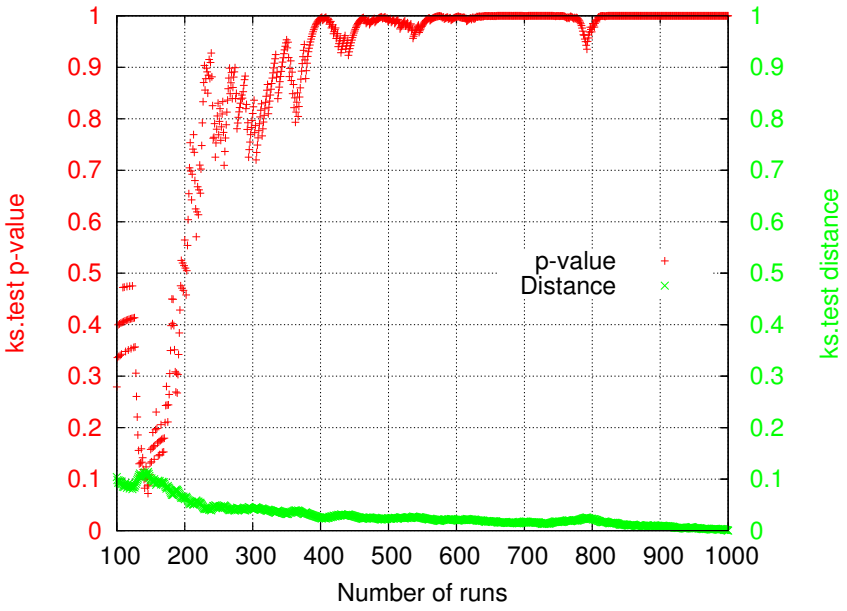

Figure 6.1: Convergence (measured as p-value and distance) of distributional properties, through Kolmogorov-Smirnov test against a distribution based on 1000 samples.

6.3 Hadoop Traffic Variations

When using Hadoop, people submit jobs to perform the data processing and may change the cluster configuration to improve performance. We first explore the traffic generated by various Hadoop jobs, starting with the structure of the generated traffic. Then, we explore a few cluster settings that usually occur in the real world. Our overall aim is to gain a general understanding of the Hadoop traffic, to enable us later to generate similar traffic.

6.3.1 Hadoop Traffic Decomposition

Using the sflow records, we start by splitting the traffic into its individual types (using the port numbers). Table 6-B presents the breakdown of traffic for each type of job. We found the traffic portion is consistent for each job given various job input datasets. From both virtual and physical clusters, the amount of control plane traffic is negligible, with

Table 6-B: Traffic composition of Hadoop jobs.

Job	**HDFS read**	**HDFS write**	**Shuffle**
sort	0.01	0.7	0.29
wordcount	0.03	0.66	0.31
TeraSort	0.1	0	0.9
Bayes	0.54	0.46	0
kmeans(itr-1)	0.31	0.67	0
kmeans(itr-2)	0.07	0.82	0
PageRank(itr-1)	0.10	0.66	0.23
PageRank(itr-2)	0.25	0.09	0.66
Hive Join	1.0	0.0	0.0
Hive Aggregation	0.98	0.02	0.0

over 99% of traffic coming from HDFS data transfers and Shuffle. Hence, control traffic can be ignored for network traffic generation, as its impact is negligible. However, for data plane traffic, the diversity across jobs is significant. For example, we observe no Shuffle traffic in map-only jobs such as Bayes and K-means compared to 90% in TeraSort (seen in Table 6-B). Moreover, the PageRank job generates just 1% of its traffic from HDFS read, compared to 54% for the Bayes job. No shuffle traffic can be seen if it is a map-only job (*e.g.,* Bayes and K-means). Further more, in cases where there are not HDFS replications set, HDFS write traffic is eliminated. Overall, it can be seen that the proportions (and amounts) of traffic generated by these three key protocols are largely based on the job type and dataset.

6.3.2 Traffic Pattern over Jobs

We now take a closer look at three specific types of jobs. We vary the input parameters and dataset for each job, to see how the traffic is affected.

6.3.2.1 Kmeans

HiBench generates various data inputs, represented by the number of samples and clusters. We run K-means with 10 clusters (k=10) and a number of samples ranging from

500k to 20m. The K-means algorithm runs recursively over multiple rounds. We separate each iteration in our comparison. As shown in Fig. 6.2, between the first and the penultimate iteration, the traffic increases linearly as a function of the data input size. The last iteration is different, as it involves only the computation of the clusters.

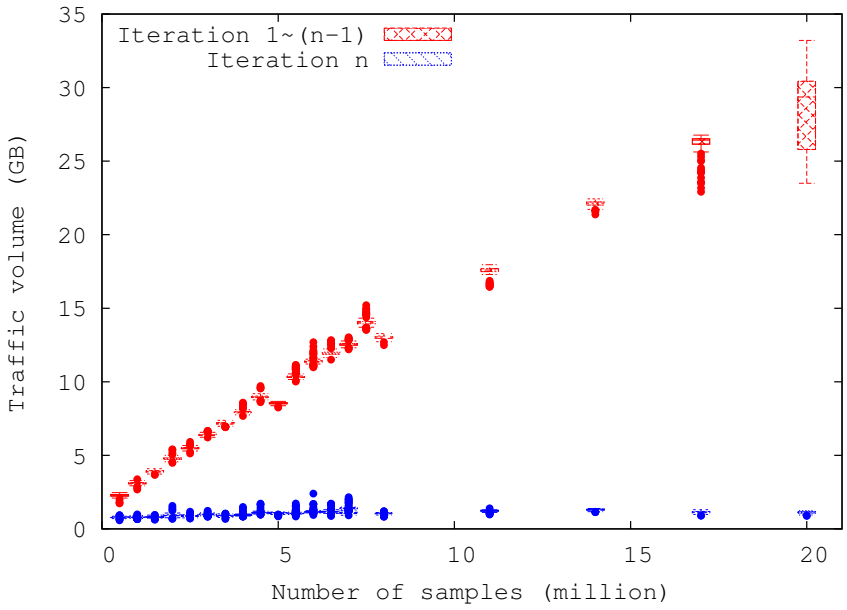

Figure 6.2: Traffic volume of kmeans over various data inputs.

From the implementation, we indeed know that all iterations but the last require computing the clusters and assigning the samples to each cluster. Therefore, more traffic will be exchanged during this assignment. The last iteration is only concerned with the computation of the clusters, a fixed volume of traffic is generated.

6.3.2.2 TeraSort

TeraSort jobs take the dataset to be sorted as a parameter. Therefore, we rely on various data files with increasing size and the degree of sorting. We use TeraGen to create various data files ranging from 1GB to 20GB, then sort the data running TeraSort while monitoring the Hadoop traffic.

The traffic volume exhibits a linear trend with respect to the data file size and the

average traffic volume observed in the network. Surprisingly though, the variability is quite high even for identical job settings (about a quarter of the file size, similar to Figure. 6.4). To better understand this, we experiment with the TeraSort ideal case (i.e., pre-sorted data), and observe the traffic in this specific situation. Even in this case, the generated traffic is as large as when the data is not pre-sorted but random. The reason lies in the fact that the data is processed in a distributed manner, and is spread out across the cluster irrespective of whether it is already sorted or not. Therefore, for the traffic seen on the network, how well the data is sorted beforehand is irrelevant.

The positive aspect, however, is that to generate realistic TeraSort traffic, only the data size needs to be considered, not how well this data is already sorted. Also, thanks to the linear trend in generated traffic as a function of data size, from a few instances of data it is possible to infer the relationship between data size and amount of traffic.

6.3.2.3 PageRank

The distributed implementation of PageRank in Hadoop adopts two stages. First, it generates partial matrix-vector multiplication results. Then, it merges the multiplication results. Both stages run recursively until the rank converges. We expect again that dataset size is linearly related to the amount of generated traffic

We run a PageRank job on datasets containing from 2 million vertices (1.1GB) to 10 million vertices (5.5GB), with a number of edges following by a Zipf 0.5 distribution over all the vertices. As with K-means and TeraSort, we found the traffic also follows a linear pattern as a function of dataset size.

6.3.2.4 Comparison Across Job Types

It is also possible to inspect how each of these traffic components is generated across the multiple stages of a job. By definition, each stage occurs sequentially: *HDFS Read*

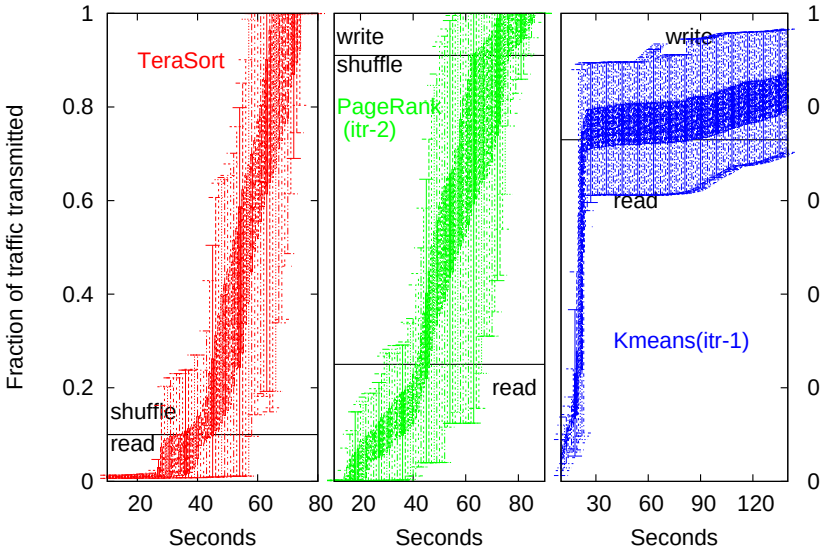

Figure 6.3: Fraction of traffic transmitted over time, for TeraSort, PageRank and Kmeans. The horizontal lines demarcate HDFS Read, Shuffle and HDFS Write traffic (Note the TeraSort is HDFS read and shuffle only, the tail of Kmeans is truncated for plotting).

loads the data onto mapper nodes, *Shuffle* exchanges data between the mapper/reducer nodes. Finally, *HDFS Write* stores the result. We focus on how these traffic components are generated in the previous three example jobs: K-means, TeraSort and PageRank. Figure 6.3 presents the amount of traffic generated per-second on the physical cluster: the second iteration of PageRank jobs (with 10 million vertices), TeraSort jobs with 6GB data, and the first iteration of K-means jobs with 14 clusters and 14 million samples per cluster. The amount at each second is shown as a box plot, with data taken from 400 runs.

It can be seen that each job exhibits the same trends of traffic in terms of the HDFS read, shuffle and HDFS write sequence, but with differences on varying ratio between the three core traffic components. For instance, TeraSort's first 10% of traffic is HDFS Read, followed by the remaining 90% that corresponds to Shuffle traffic. Similarly, for PageRank, there is 25% HDFS Read at the beginning, then 66% Shuffle, and finally the

last 9% is HDFS Write. All traffic seen on the network follows this three stage sequence. Clearly, this stability in the structure gives a clear guidance that HDFS read, shuffle, and HDFS write traffic need to be investigated separately to get a precise reproduction of the Hadoop traffic. Importantly, this must be performed on a *per-job* basis to reflect the diversity observed.

6.3.3 Traffic Pattern over Cluster Settings

The previous section has shown that the bulk of the traffic is generated by HDFS and shuffle (with the majority from HDFS export) for the considered jobs. Next, we explore how these traffic volumes change when we change the cluster settings: data distribution, HDFS replication rate and cluster size, as those are parameters often set first when initiate a Hadoop cluster. Note that we do not consider settings such as Java virtual machine memory allocation that usually affect the computation performance rather than networking [142], but leave it as further work.

6.3.3.1 Data Distribution

Hadoop tries to allocate computation to mappers based on the data that is already stored on that particular mapper (note that mapper nodes also participate in HDFS). To confirm the effect that the pre-existing data distribution has on Hadoop traffic volumes, we repeat our TeraSort jobs with a variety of placements. We generate a variety of datasets ranging from 100MB to 1.9GB (in steps of 200MB) using the default TeraGen benchmarking tool. We then use the Terasort job to sort the dataset ten times. We find that for 59 out of 90 jobs, the mapper is running on the node without any data stored, thus the a huge HDFS read traffic. Although Hadoop tries to assign the mapper to the Node Manager where the map split is stored, the data node can be shared by multiple splits. Thus, sometimes the mapper has to choose other nodes to run the splits.

A more specific investigating found that, although the data should be equally spread

between nodes, two data nodes may hold three times more blocks than others in the cluster. Indeed, TeraGen always assigns data to the reducer nodes.

HDFS import traffic is driven by the effectiveness of the data placement. To confirm this, we inject the data into HDFS by copying it externally with the"fs -put" command. We witness a 27% decrease in HDFS traffic. Note that this traffic only accounts for 2% of the total job traffic.

6.3.3.2 Replication Settings

HDFS stores data on nodes across the cluster. These nodes may also be used as mappers for computational tasks; hence, a mapper that already contains the data locally generates no extra traffic. To improve resilience, HDFS has a replication set that allows the same block to be stored on multiple nodes. Intuitively, the higher the replication level, the lower the traffic volume.

To explore the effect of replication, we run a new set of TeraSort jobs (of 4GB), varying the replication levels. Figure 6.4 shows that 86% of the traffic goes away when increasing the replication rate from 1 to 2. A replication level of 3 further decreases it by 56%; and by another 23% for 4, but barely changes after that. Clearly, additional replication suffers from diminishing returns, as beyond a level of 3 benefits become insignificant. For example, in our cluster of 15 nodes, with the maximum replication of 15 and file's replication is located on each of the node, no HDFS import traffic is seen. Note that the savings in HDFS import come with a startup cost, as it is necessary to preload the data, which may be costly in a Cloud platform.

We also run the same experiment with PageRank and K-means jobs. We again find that HDFS input traffic can be reduced with an increased number of replications. However, the HDFS output traffic increases, while the shuffle barely changes. This is because shuffle is used for exchanging newly created data during a job. 90% of the traffic is the shuffle in TeraSort. Moreover, sample traces from Facebook [236] indicate that

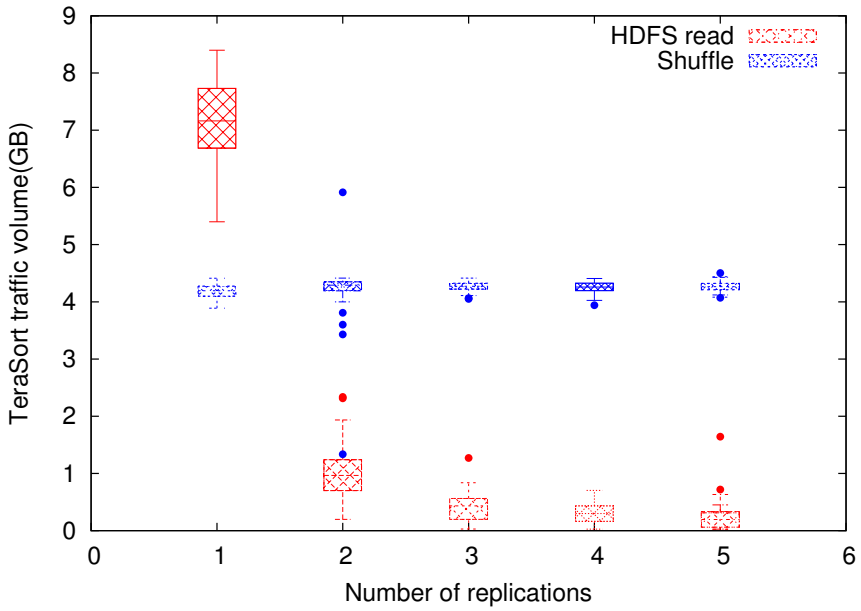

Figure 6.4: Volume of HDFS import and shuffle traffic for TeraSort 4GB jobs. Increasing HDFS replications in the cluster does not necessary decrease HDFS traffic.

58% of the total traffic is the shuffle.

6.3.3.3 Cluster Size

The final parameter we consider is the size of the cluster (i.e., the number of nodes in the cluster). Cluster size may impact network traffic significantly, as nodes will need to hold more blocks if fewer nodes are available. Thus, we now study the relationship between traffic and number of nodes.

We run TeraSort over a 1GB dataset while varying the cluster size. Overall, the total traffic volume follows the same distribution, though the traffic generated per node is decreasing from 6 to 10 nodes, and remains constant afterwards. This is because Hadoop assigns to each mapper units of data called splits. Therefore, for a fixed input data size, the number of mappers, as well as nodes needed, will be fixed. For instance, a 1GB TeraSort job will utilise a maximum of 8 nodes. This presents an advantageous property: the traffic for TeraSort 1GB jobs on a 15 node cluster will be identical to

that of any other cluster size above 10. We further confirmed this on other jobs such as PageRank and K-means. This property enables us to predict the generated traffic, once we know the cluster capacity.

6.3.4 Summary of Hadoop Traffic

In this section, we have explored some aspects of Hadoop traffic. We have seen that these patterns have some predictable trends, e.g., with a linear relationship over job parameters, which will enable us to model the Hadoop traffic under common cluster settings.

The distributed nature of the Hadoop computation requires the data to be distributed across different machines. In Hadoop, the data delivery is driven by the dataset and algorithm which, generates traffic in a unique manner. We can capture the distribution of HDFS and shuffle traffic for different types of jobs, as long as we have at least 400 job runs (as explained in Fig. 6.1). Furthermore, we quantify how the job parameters and cluster settings linearly affect the traffic.

6.4 Modelling Hadoop Traffic for Replay

In this section, we start with details of how we capture the Hadoop traffic and extract the necessary distributions. First under default settings and then using various jobs, job settings and cluster settings. The purpose is to then prime our later re-generation of synthetic Hadoop traffic in Chapter 7.

6.4.1 Traffic Parameters

After the traffic is collected, we aim to profile the traffic at the *application level*. Normally, traffic can be described at the network level by using the five-tuples: the protocol,

source/destination address and source/destination port. However, to fully capture the Hadoop traffic along with its behaviour, we need to use extra parameters to take the application level behaviours into consideration.

Thus, we propose a new set of parameters in an attempt to best capture Hadoop traffic, agnostic to the underlying network features (*e.g.,* transport protocol, topology). For each traffic component (HDFS read, write and shuffle), the traffic is described by five parameters: the number of containers, number of source nodes per container, number of flows per source, flow size and flow start time.

We chose this approach because the 5-tuple is the minimum required to identify flow-level behaviour. As we wished to achieve high scalability, we wanted to operate on this flow-level to avoid the complexities and overheads introduced by packet-level models. The model is therefore the most lightweight data structure for recording traffic. We will now describe the 5 elements of the tuple, and their role in traffic generation. To aggregate flows into a single model that could be used for traffic replay, we recorded the number of containers, number of source nodes per container and number of flows per source. This is sufficient for regenerating the number of flows in the simulator. However, it does not capture the size or timings of the flow. Hence, it was necessary to supplement these three parameters with flow size and flow start times. These container oriented traffic parameters can help us summarise the traffic to each of the containers Hadoop generated, without constrains from the type of job. Thus giving us accuracy when replaying the traffic. The same set of parameters are captured for each component including HDFS read, shuffle, and HDFS write.

Following this, we then extract the full distribution of parameter values (*e.g.,* flow sizes) across a number of job runs (400 by default), *i.e.,* we repeat the same job 400 times. This is necessary because the values of the parameters are non-deterministic and vary per-run. Our experiments show that 400 runs give us enough samples to construct the distribution 6.2. And by quantifying the parameters via a distribution, we can later compare the parameters from various settings and even extrapolate the parameters in

the future.

6.4.2 Extracting the Traffic Distributions

The previous section shows how we capture traffic from a live cluster. We next explore how to model the Hadoop traffic, so that it can be later replayed within a simulator. Here, we run TeraSort, PageRank, K-means and Hive join with the default cluster settings listed in Table 6-A. We emphasise that our methodology works across *any* job; we use these four simply as exemplars.

The approach taken to traffic modelling is straightforward. We collect the empirical traffic over 400 runs and, for each run, capture the parameters: the number of containers, number of source nodes per container, number of flows per source, flow size and flow start time. We then perform a maximum-likelihood fitting of univariate distributions, evaluate the fitting by using the KolmogorovSmirnov test (ks.test) and then select the distribution with the lowest distance found after passed the critical values.

To show the fitting results, we present the CDF of empirical samples (red) and fitted value (green) for each of the parameter that we used to describe the Hadoop traffic including number of HDFS containers (Figure 6.5), number of source nodes per container (Figure 6.6), number of destinations per source node (Figure 6.7), number of flows per source-destination pair (Figure 6.8), HDFS flow start time (Figure 6.9); Number of shuffle containers (Figure 6.10), number of source per shuffle (Figure 6.11), number of destination per shuffle (Figure 6.12), number of shuffle flows (Figure 6.13), shuffle flow start time (Figure 6.14); HDFS write containers (Figure 6.15), number of HDFS write traffic source nodes (Figure 6.16), number of HDFS write destinations (Figure 6.17), number of HDFS write flows (Figure 6.18) and finally HDFS flow start time (Figure 6.19). The shuffle part has only TeraSort and PageRank as K-means, as Hive jobs do not have shuffle traffic. These plots give us an intuitive visualization of the distribution fitting result.

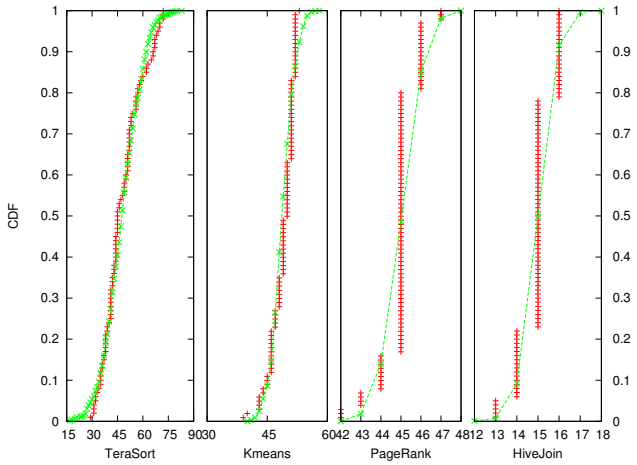

Figure 6.5: Number of mapper containers.

For the distributions found, we can see that for the number of containers, number of sources/destinations per containers, number of flows and flow starting time all have one thing in common: they are well fitted by the Normal distribution. However, for the HDFS flow size, we find two other types of distributions that fit different jobs: for mappers requiring the data from one block only (*e.g.,* TeraSort, K-means and PageRank) the HDFS read flow size is binary; for mappers that require the data from more than one block (*e.g.,* Hive, join) the fitted distribution is Weibull. A script is used to automatically extract these parametrized distributions using its maximum-likelihood fitting.

Apart from the visualisation, we list the numerical distribution parameters found for various TeraSort jobs with different job settings in Table 6-C. This includes the mean and standard deviation for the Normal distribution, the probability for Bernoulli or shape and scale for the Weibull. We can see that each of the parameter can be fitted by a distribution accordingly. The fitted values are also increased linearly correlated to the TeraSort file size. This table is the core output of the traffic modelling, and will be used as the input for further generation of the traffic which will be illustrated in the next chapter.

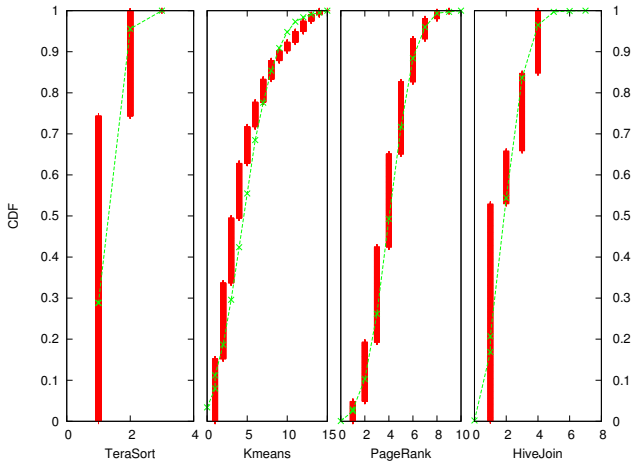

Figure 6.6: Number of sources per mapper.

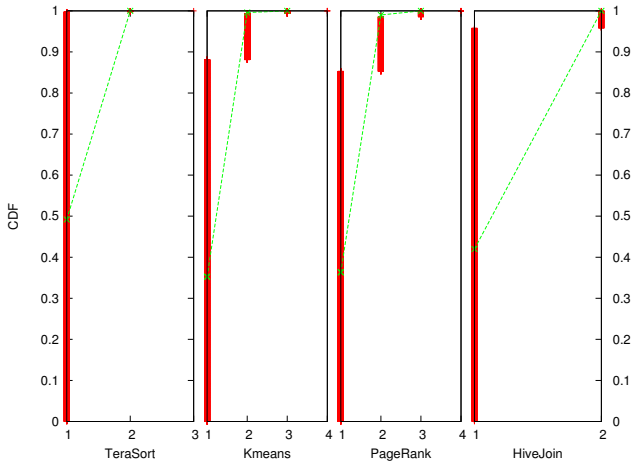

Figure 6.7: Number of mapper flows.

Table 6-C: TeraSort traffic distribution.

	HDFS read				Shuffle										HDFS write							
TeraSort file size	number of containers		start time		number of containers		number of src		number of flows per src		size per flow(MB)		start time of flows		number of src		number of dst per src		number of flows		start time	
1.00	7.36	0.57	12.90	1.76	1.56	0.50	3.96	1.70	1.00	0.00	22.42	10.34	21.98	1.11	7.50	0.98	1.90	1.23	1.08	0.29	27.62	1.80
3.00	16.33	2.85	14.75	4.28	1.97	0.17	8.49	2.30	1.11	0.32	21.03	11.99	25.60	3.33	12.72	1.11	2.84	2.51	1.25	0.56	40.13	4.80
5.00	24.92	6.02	16.91	7.02	2.00	0.14	11.17	2.53	1.28	0.53	21.63	13.53	30.98	4.94	14.12	0.87	3.71	3.13	1.41	0.77	55.68	9.17
7.00	34.42	11.00	18.65	8.40	2.00	0.00	12.80	1.55	1.51	0.78	21.45	12.79	35.72	6.44	14.33	0.76	4.40	3.40	1.59	1.01	71.73	15.79
9.00	42.82	13.80	20.14	12.78	2.00	0.00	13.25	1.17	1.69	0.97	23.43	14.14	40.09	7.79	14.82	0.41	5.00	3.58	1.73	1.18	86.10	20.11
11.00	44.45	20.51	22.54	16.57	1.99	0.10	13.34	1.30	1.86	1.13	21.58	13.88	45.13	9.47	14.92	0.27	5.93	3.34	1.84	1.36	104.11	27.00
13.00	52.38	24.25	27.38	21.89	2.00	0.00	13.35	1.50	2.08	1.43	23.61	14.14	51.22	11.67	14.98	0.14	6.56	3.27	1.96	1.54	121.88	32.00
15.00	57.95	28.02	29.89	21.28	2.01	0.10	13.44	1.60	2.22	1.55	23.00	11.95	57.16	14.50	15.00	0.00	7.12	3.12	2.08	1.70	136.58	36.04
17.00	56.97	29.77	32.62	25.54	2.01	0.10	13.62	1.52	2.32	1.74	23.18	12.84	62.47	14.75	14.99	0.10	7.62	2.98	2.21	1.88	163.96	47.80
19.00	70.29	36.61	28.95	29.59	2.00	0.00	13.78	0.84	2.56	1.90	23.02	12.45	64.12	15.74	15.00	0.00	8.07	2.89	2.33	2.07	191.03	65.12
21.00	81.77	43.43	28.97	28.43	2.00	0.00	13.80	0.76	2.62	2.07	23.52	12.64	69.17	17.65	15.00	0.00	8.56	2.68	2.44	2.22	221.11	77.13

6.4.3 Modelling Impact of Cluster Settings

The previous subsection has shown the distribution of traffic metrics across a variety of jobs. Importantly, it has used simple best-fitting to extract the probability distri-

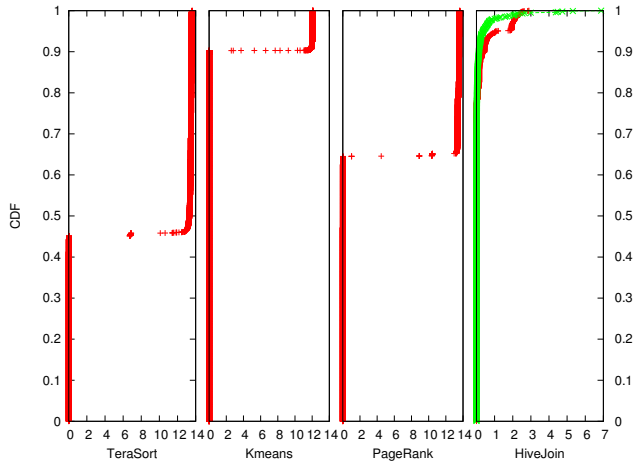

Figure 6.8: HDFS read flow size(10MB).

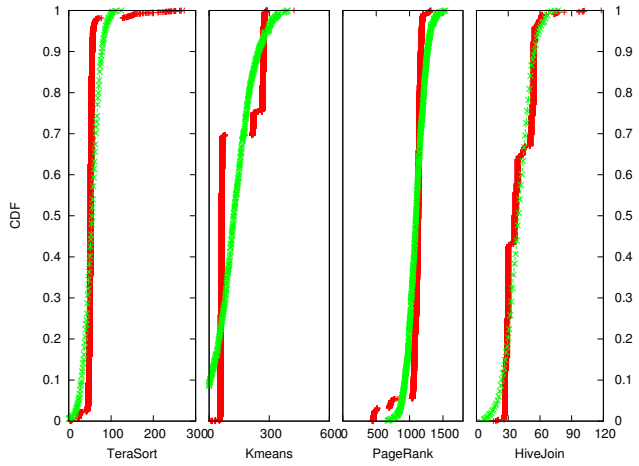

Figure 6.9: HDFS read flow starting time.

butions of these metrics to later allow replay. However, we have also observed that individual cluster settings with Hadoop can heavily impact these values. Hence, it is necessary to capture the impact that these different cluster settings have on the traffic makeup. To achieve this, we repeat the same methodology while varying several key cluster parameters; under each parameter setting, we extract the five-tuple model. This is then embedded within the parameter file allowing an experimenter to select the cluster settings they wish to generate traffic for.

We start with a set of parameters that are commonly used in practice to tune cluster performance, namely data replication, cluster size, block size, input split size, number

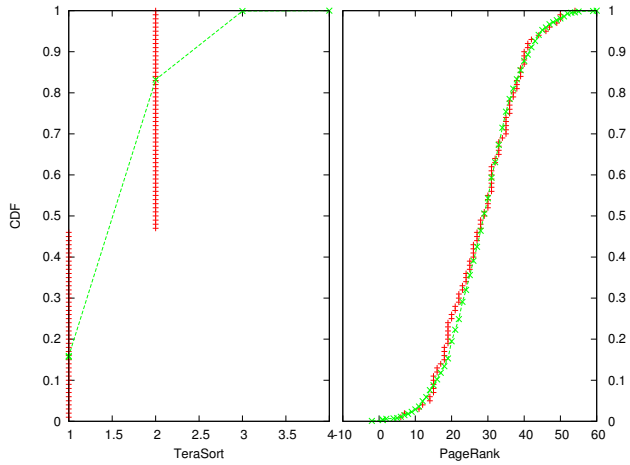

Figure 6.10: Number of reducer containers. Although the low number of data points makes the fitting difficult, we will ground the fitted value to simulate the discrete value.

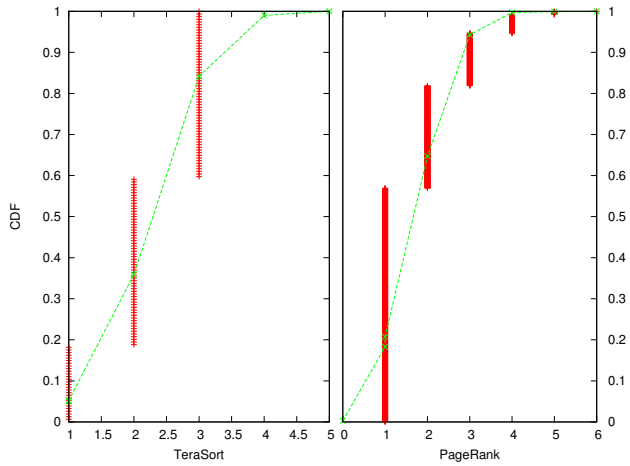

Figure 6.11: Number of sources per reducer.

of reducers and output replications. Clearly, it is not feasible to fully explore this large sample space by executing millions of jobs. Thus, we use the default settings as a baseline (as shown in Table 6-A), and change only one setting at a time to explore the difference. As well as executing these tests to extract the models for the parameter file, the following section will explore the impact that differing settings have to elucidate the nature of Hadoop traffic better.

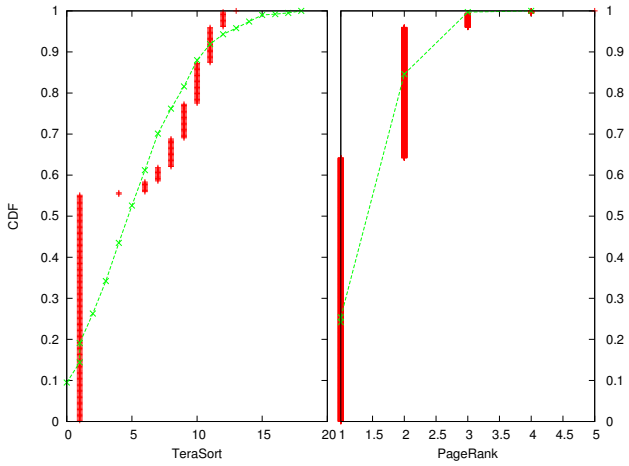

Figure 6.12: Number of flows per reducer source.

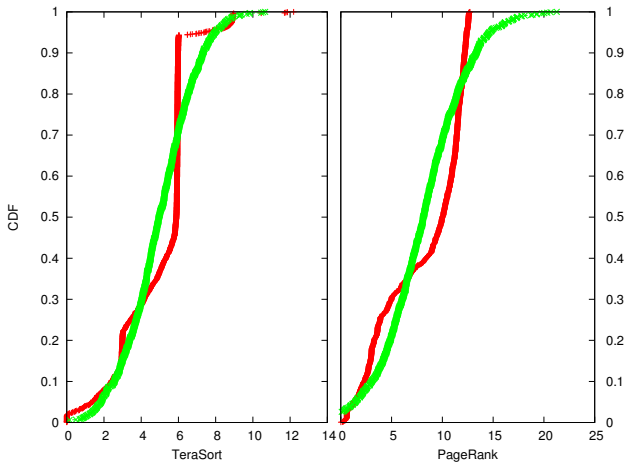

Figure 6.13: Shuffle flow size(10MB).

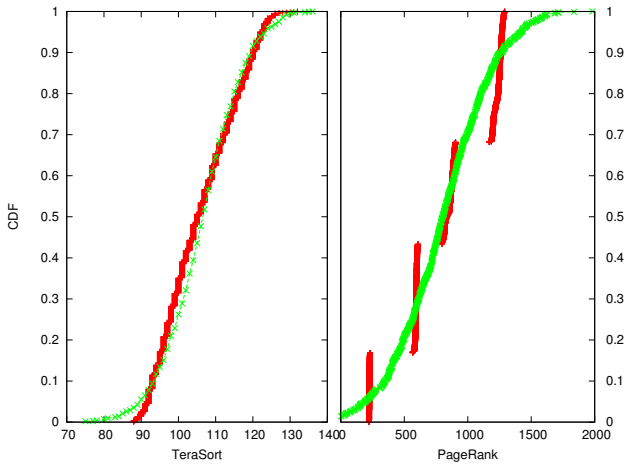

Figure 6.14: Shuffle flow starting time.

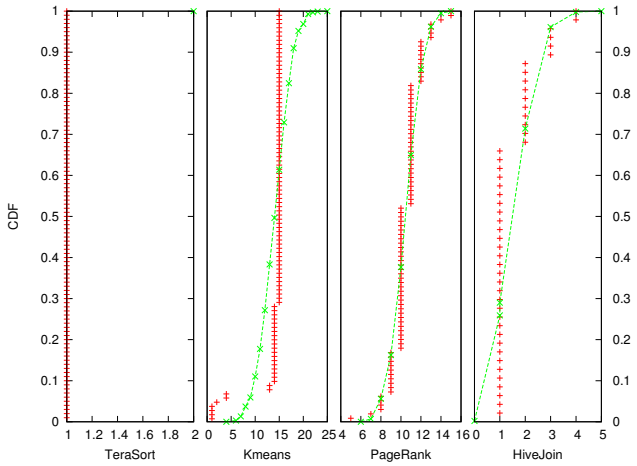

Figure 6.15: Number of HDFS write source.

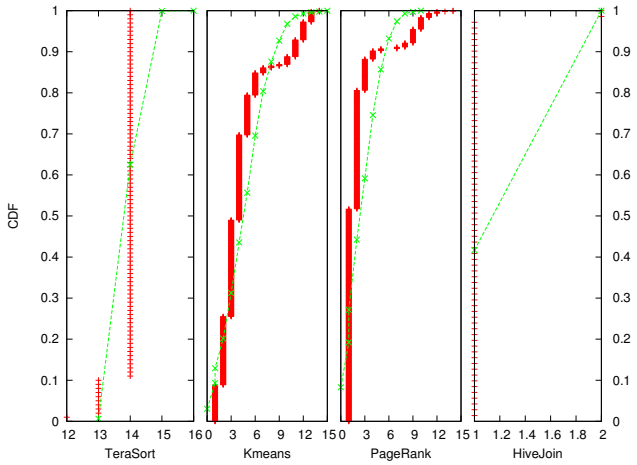

Figure 6.16: Number of HDFS write destination per source.

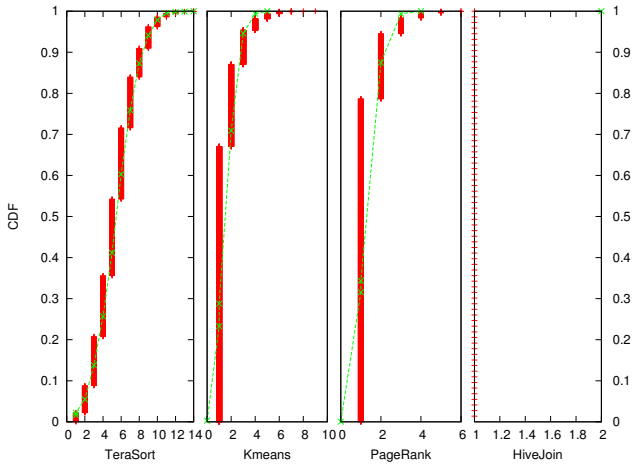

Figure 6.17: Number of flows per HDFS write source destination.

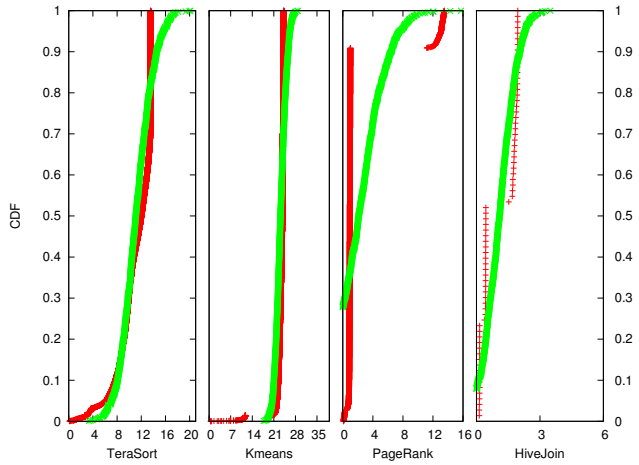

Figure 6.18: HDFS write flow size(10MB).

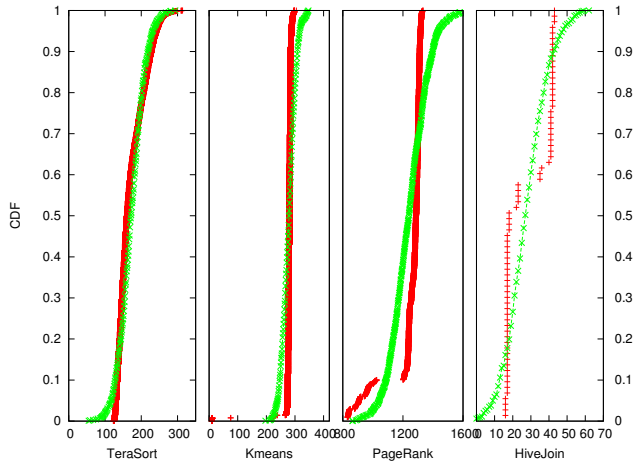

Figure 6.19: HDFS write flows starting time.

6.4.3.1 Replication and Cluster Size

Amongst the most important decisions made when operating a Hadoop cluster are the *replication rate* (*i.e.,* number of data replicas) and *cluster size* (*i.e.,* number of nodes). When data is stored in HDFS, the data is split into file blocks. Each block may be stored on multiple nodes for fault tolerance; this is controlled by the replication setting. With more replicas (or fewer work nodes in the cluster), each node will clearly have to store a greater number of blocks. As we have discussed previously, the mapper reads a block at the beginning of execution, and traffic will be generated if the block is not locally stored.

In essence, the above can be modelled using the probability by which a mapper will have a local copy of the required block. If a local copy is not available, traffic will be generated. To understand this probability, we have extracted the algorithm used for replica management from the Hadoop source code. We find that blocks are randomly assigned to nodes within the cluster. Following this, when a mapper container is assigned to a node, it is, by preference, allocated to the node where the data is already stored. However, this does not always happen.

Rather than exploring the impact of replication by executing real jobs, we simply use this (deterministic) assignment algorithm, which produces the probability by which a mapper will be co-located with its data. To compute this, it takes four input parameters: number of nodes, number of blocks, number of replications and number of mappers. The algorithm first randomly assigns blocks on each node. Note the total number of blocks will be the number of blocks input multiplied by the number of replicas. The algorithm then starts to assign the mappers to each node. It then simply counts the number of mappers running on a node without the requested block stored. Finally, it outputs the counted number divided by the total number of mappers: this is the probability of having traffic generated during the mapper process. We find that the empirical probability that a mapper reads a remote block follows the Bernoulli distribution (parameters listed in Table 6-C).

To confirm the above is correct, we compare our statistic results against the behaviour of real Hadoop cluster. To save time on running jobs again and again, we first use an emulation approach to get a precise estimation[4]. The emulation process set a number of nodes and a number of data blocks; then place each block randomly onto each node. When the job starts, the mapper will always prefer a node with data stored. However, some mapper may end with fetch data remotely as the nodes with data stored already have mapper running on. We record the number of those mappers that need to fetch data and calculate the statistic mean over a thousand emulation runs.

[4]Code can be found in https://github.com/deng113jie/keddah/locationemu

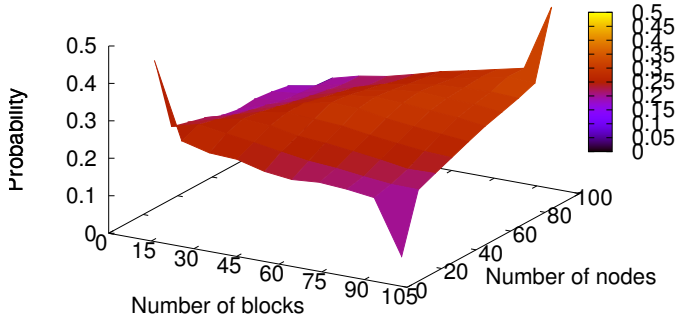

Figure 6.20: The probability that a mapper requires a block not locally stored, with various number of nodes presented and number of blocks required. Cluster replication set as 1.

We present the values found for a various numbers of blocks and numbers of nodes combinations under different replication settings, including 1 replication (Figure.6.20), 2 replications (Figure.6.21) and 3 replications (Figure.6.22). We can see that the probability can be very low when the number of blocks exceeds the number of nodes (hence a block will be stored on the node) or the number of blocks is less than the number of nodes (thus the mapper is running on the node with the block stored); otherwise the probability can be rather high when the number of blocks is similar to the number of nodes (hence the mapper happens to run on a node without the block stored). Not surprisingly, we found the same result from running empirical traces in our real cluster. We can see that the probability varies over different combinations but also decreases with more replication set in general.

6.4.3.2 Block Size and Input Split

The block size decides how many blocks each file is fragmented into when stored in HDFS. The input split size is the data file size that each mapper processes. By default,

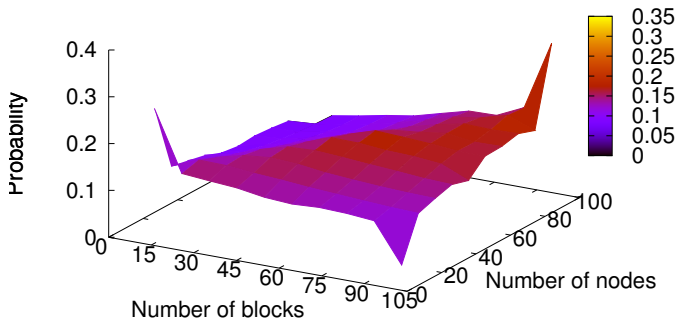

Figure 6.21: The probability that a mapper requires a block not locally stored, with various number of nodes presented and number of blocks required. Cluster replication set as 2.

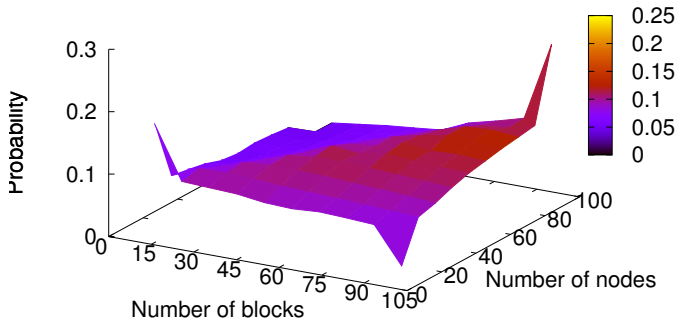

Figure 6.22: The probability that a mapper requires a block not locally stored, with various number of nodes presented and number of blocks required. Cluster replication set as 3.

the block size is treated as the input split size. Next, we run jobs over various block and input split sizes, and see how the traffic is affected. Each run results in a new 5-tuple model, which is included in the parameter file (allowing people wishing to replay the traffic to select their preferred block and input size).

To explore how these parameters impact traffic generation, Figure 6.23 plots the number of map containers used when experimenting with various block sizes. Although

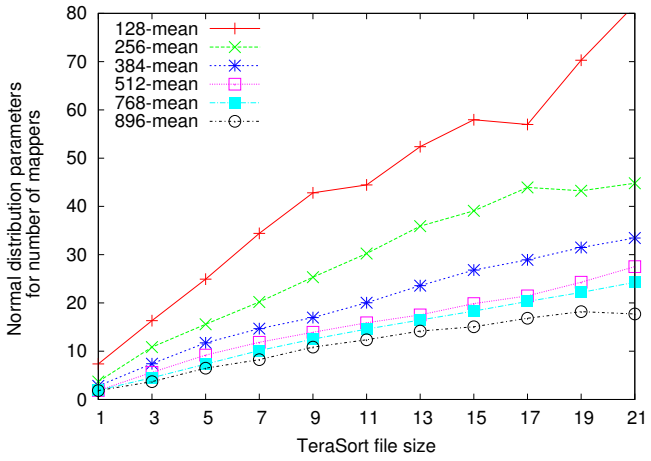

Figure 6.23: Number of map containers found over various block sizes.

various values are found from 400 runs, the Y-axis shows the mean value of distribution for comparison purposes. When we alter the block size or input split size, the number of mappers decreases linearly. This is expected as the change of block size will reduce the number of blocks stored. Thus the mapper has a higher chance of running on a node with blocks locally stored. This linear relationship found can further be used to infer traffic generation parameters in the next chapter.

6.4.3.3 Number of Reducers

Another key metric is the number of reducers, which will clearly affect the reduce phase of MapReduce. The number of reducers can be controlled with mapreduce.job.reduces setting. Hence, we vary this parameter and generate the 5-tuple model for each parameter setup. We find that the number of reducers is affected by such configuration when running TeraSort, PageRank and Hive join jobs. Surprisingly, this is not always the case though. Figure 6.24 presents the fitted mean value found for the number of reducers from the 400 empirical runs against various TeraSort job parameters. It shows that the number of reducers increases with the setting, but not when the input data file size is small. For example, even though it is set to 3, only one reducer is started when running TeraSort 1G job. Hence, the mapreduce.job.reduces setting constitutes a limit,

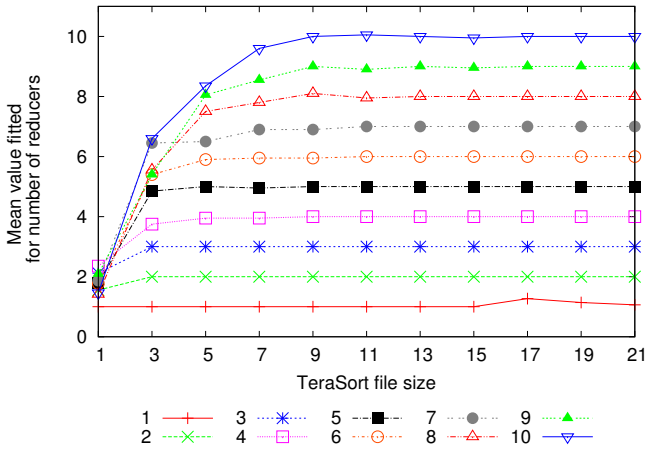

Figure 6.24: The number of reducer containers found, varies over the number of reducers setting.

not a lower-bound. By executing and capturing jobs with these different settings, we automatically capture the number of reducers seen in this parameter file.

6.4.3.4 Output Replication

The final parameter we inspect is the output replication level; this dictates the number of replicas stored for the output of the job. Clearly, this will affect the amount of HDFS write traffic. Again, we execute a number of jobs whilst varying the replication setting. In each case, we record and model the traffic for storage within the parameter file.

Figure 6.25 and Figure 6.26 show the variations by presenting the mean value fitted for the number of HDFS write traffic sources and number of HDFS write traffic destinations per source. We find that this increases the number of sources and destinations, since increasing the replica degree by one will naturally result in one more source-destination pair. That said, the distribution of other parameters such as flows per source-destination pair does not change. Again, we can see that the value increases linearly with respect to the total number of nodes in the cluster.

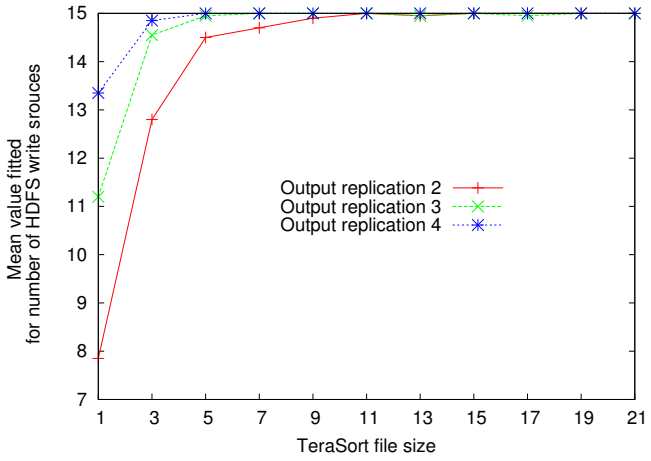

Figure 6.25: The number of HDFS write traffic sources affected by various output replications.

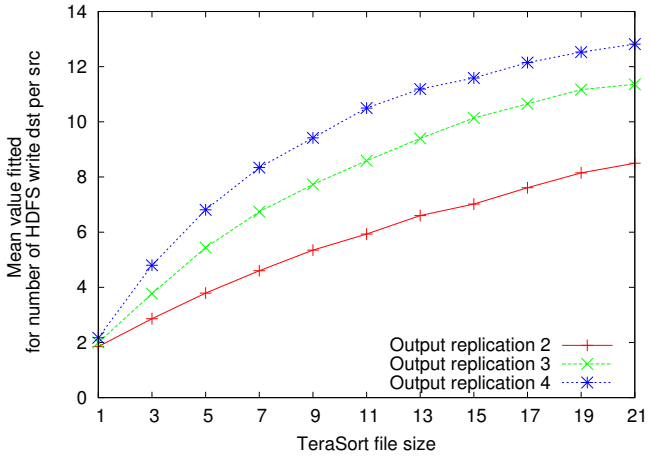

Figure 6.26: The number of HDFS write traffic destinations affected by various output replications.

6.4.4 Review of Modelling Hadoop Traffic

In this section, we have empirically captured, modelled and explored the nature of Hadoop traffic. The purpose behind this has been to underpin the recreation of synthetic Hadoop traffic. Using a variety of jobs and Hadoop parameters, we have quantified the resulting traffic that is generated. Those jobs are carefully chosen from text searching, sorting, machine learning, graph processing and database joins. This extensive list of operations can help fully expose Hadoop network traffic behaviours. In fact,

those jobs are widely used in Hadoop performance benchmarking, in TPCx-HS [146], HiBench [148], SWIM [236] and Puma [237]. The jobs are also often used in Hadoop performance evaluations in the research community [144, 238–241]. Apart from the jobs, we also investigated a few cluster settings that are often tuned in practise: cluster size, replication setting, block size, input split size, number of reducers and output replications. Those settings are very basic settings that everyone needs to configure on their own cluster [242] yet, no study has been made before on their effect on the network performance of Hadoop. We found out, indeed, those parameters can explicitly impact the network behaviour of Hadoop.

For each parameter under consideration, we have investigated the distribution of its values across 400 runs to capture its properties. In total, we use 15 distribution parameters to fully capture the Hadoop traffic. So that the traffic of Hadoop can be quantified for comparison and even reproduced. We further extended the investigation to a set of benchmarking jobs, jobs settings and cluster settings. We found the linear correlation between the settings with the traffic generated. On the one hand, the accuracy of the traffic distribution is validated by the linearity, on the other hand, the linear correlation can help infer the traffic distribution.

Regarding the reproducibility, though we use the most straightforward topology to observe the traffic, the result should apply to other sophisticated typologies as well. We aim to capture the traffic behaviour from the application level so that the behaviour is not affected by network layer components, such as topology. Thus the number of traffic sources, number of flows and flow size, etc. are all independent of the network topology. However, to reproduce the traffic, the flow start time relies on the network components, e.g., topology, bandwidth. Still, others can use the Keddah tool to generate the traffic statistics from scratch instead of using the result we published.

6.5 Summary

This chapter has characterised and modelled the traffic of Hadoop jobs. With the goal of generating the Hadoop traffic in the simulation environment, understanding the traffic is the first step. By characterising the traffic under various jobs, job setting and cluster settings, we have revealed the traffic generated under different scenarios. Also, a linear correlation between the traffic distribution with the job parameters can be established according to the traffic. With such understanding, we further proposed 15 distributions traffic parameters that used to capture the Hadoop traffic with Yarn containers behaviour included. By using the 5-tuple on each HDFS read, shuffle and HDFS write, the Hadoop traffic can be described, compared and further reproduced. In the next chapter, we will give a detailed explanation of the traffic reproduction. We adopt a simulation approach to evaluate network performance under Hadoop traffic.

Chapter 7

Reproducing Hadoop Traffic for Profiling Network Performance

7.1 Introduction

In the previous chapter, we have characterised and modelled the traffic of Hadoop. As a sophisticated distributed processing application, the traffic of Hadoop is driven by the processing job, with identical traffic patterns of HDFS read, shuffle and HDFS write. Though we found the network hugely impacts the distributed processing performance by transmitting the traffic, we still lack in efficient method to evaluate the network design performance. Given Big Data applications is one of the main contributors to intra-datacentre traffic, this lack in efficient method prevents researchers from making network innovations on the network for distributed processing applications.

Thus, in this chapter, we present an open source tool that allows network researchers to evaluate their data centre-specific and Hadoop-specific networking studies in a *simulated* fashion. As part of this, the work done in the previous chapter can help us reproduce the realistic traffic in the simulated environment. Based on the Hadoop traffic characterised and 15-tuple argument proposed previously, in this chapter, we present *Keddah*,

a tool that allows (*i*) Hadoop traffic to be captured within real clusters; (*ii*) anonymous traffic models to be compiled; and (*iii*) the traffic to be replayed in simulated environments. Collectively, this allows any researcher to evaluate their network-level innovations with realistic traffic patterns.

Based on the traffic models, Keddah generates traffic by simulating the complete execution flow between compute nodes, and the traffic exchanged between them. Keddah models Hadoop traffic at a flow level including the traffic volume, nodes involved and sessions for each protocol. We have built Keddah as an open source tool[1] on top of a network simulator (ns3), and integrated publicly available Big Data workloads [243] for network research purposes. We have included various models for immediate use, but also allow other Hadoop operators use Keddah to capture and share their traffic patterns.

This chapter is organised as follows:

- Section 7.2 outlines the architecture of Keddah and specifically illustrate each component in the later sections;
- Section 7.3 presents the implementation of Keddah, as well as how it is integrated with ns3;
- Section 7.4 discusses our validation experiments where we compare Keddah's generated traffic with workloads captured in real environments;
- Section 7.5 presents and evaluates two example applications of Keddah for data centre network research. We show the effect of multiple data centre topologies and transport-level protocols in the performance of Hadoop jobs.
- Section 7.6 concludes this chapter.

[1] github.com/deng113jie/keddah

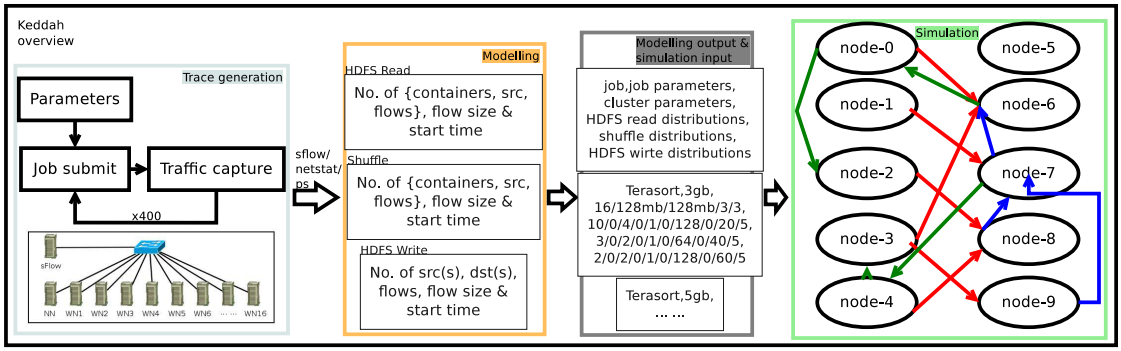

Figure 7.1: Architecture of Keddah, starting with execution of real jobs and ending in the generation of the traffic in a simulator.

7.2 Keddah Architecture Overview

This chapter exploits the traffic observations from the previous chapter to build a tool that can capture and replay Hadoop traffic. *Keddah* offers a full workflow that allows users to (*i*) capture traffic from a real cluster; (*ii*) extract the traffic distribution; (*iii*) replay it within a simulation environment. Importantly, this does not involve exporting real traffic traces — instead, Keddah generates compact traffic models that represent the traffic seen.

The workflow is shown in Figure 7.1. On the left, traffic is captured from a real cluster (as shown in Section 6.4). Following this, the empirical traffic distributions are computed (as shown in Section 6.4.2). The workflow then exports the distributions into a separate *Keddah file* that can be used to replay the traffic in a simulation environment (as shown in Section 7.3). Another key goal of this workflow is to allow *any* Hadoop operator to capture their traffic and share it with the community for replay. Importantly, by converting traffic into the appropriate distributions, we ensure anonymity and minimise exposure of sensitive information regarding a particular Hadoop operator.

The first stage in the Keddah workflow is the capturing and modelling of Hadoop traffic from real environments. The specific process is shown in Section 6.4 that, modelling the traffic and output the traffic distribution. To further automate the process, a script is used to set the cluster parameters, run the Hadoop job, capture the traffic,

Table 7-A: Keddah file header.

cluster_size , replication , block_size , input_split , reducers , no_of_mappers , no_of_src_per_mapper , no_of_flows , flow_size , hdfs_read_starttime , no_of_reducers , no_of_src , no_of_dst , shuffle_size , shuffle_starttime , no_of_src , no_of_dst , no_of_flow , flow_size , hdfs_write_starttime

modelling the traffic and output the traffic distribution. This is then used to generate a parameter file named *Keddah file*: a comma-separated values (CSV) format configuration file containing 5-tuple descriptions of each traffic type observed (HDFS Read, Shuffle, HDFS write). The full list of parameters of Keddah file is shown in Table. 7-A. This set of 20 parameters is stored for each cluster setting that has been recorded, *e.g.,* for each replication level, with the first 5 parameters are job related metadata and the last 15 are traffic distributions extracted from methods explained in the last chapter. By doing this, the Keddah file can be shared with experimenters, who wish to replay the same traffic in a simulated environment.

Before we dive into Keddah implementation, we'd like to clarify a few assumptions we made on reproducing the traffic. In terms of the hardware we assume Hadoop is running on a common cluster where the worker nodes have the same hardware setup. Parameter wise, we assume Hadoop is running with default Hadoop settings (listed in Table 6-A). As Keddah primarily performs traffic replay-like functionality, we assume that the simulated cluster is broadly similar to the live cluster where the initial traffic traces were captured. For tractability, when capturing the traffic, we only focus on the parameters that people often change when initialising their cluster, such as block size, number of reducers, etc. We did not take some system level parameters into consideration, for example cluster fault tolerance, Java virtual machine parameters, etc.

That said, it is possible to record various settings in the live cluster and replay them in the simulator - we make our simulator open source to allow others to experiment with these possibilities. The rest of this chapter will outline the various components of this workflow starting from the simulation implementation.

7.3 Simulating Hadoop Traffic using Keddah: An Implementation Perspective

The previous section has captured and computed empirical models for the distribution of various key traffic metrics. These are embedded within a Keddah file that can be publicly shared with others wishing to replay the traffic. In this section, we implement the simulation part of Keddah, which uses a Keddah file to replay traffic within a simulated environment. Keddah is built on top of ns3 so that the traffic can be reproduced while experimenting with different network possibilities (*e.g.,* topology, routing protocols). Note that this is limited to explorations *below* the application layer, such that the application layer behaviour does not vary. For example, the traffic generator can be added onto various network topologies to test the network performance. However, the traffic generator can not be used to simulate Hadoop system behaviour such as job completion time, as the Hadoop system behaviour is not the object of this work.

7.3.1 Design and Implementation of Keddah

7.3.1.1 Design Overview

The previous chapter has detailed the first stage in the Keddah workflow (*cf.* Figure 7.1), where real traffic is captured and modelled to create a Keddah file. The second stage is, therefore, the use of this file to *replay* traffic in a simulated environment. Thus, Keddah also consists of a series of new ns3 software classes that represent the entire set of simulated Hadoop functionality, *e.g.,* mappers, reducers and the Yarn scheduler. Although we have chosen ns3 due to its wide availability, Keddah could be implemented in any open source simulator. Figure 7.2 shows the class diagram of the Keddah implementation. The Keddah ns3 plug-in has three main components that represent the cluster topology and end hosts (green and blue), the Hadoop scheduler (black) and the specific Hadoop jobs (orange). Each component consists of multiple classes, represented by their

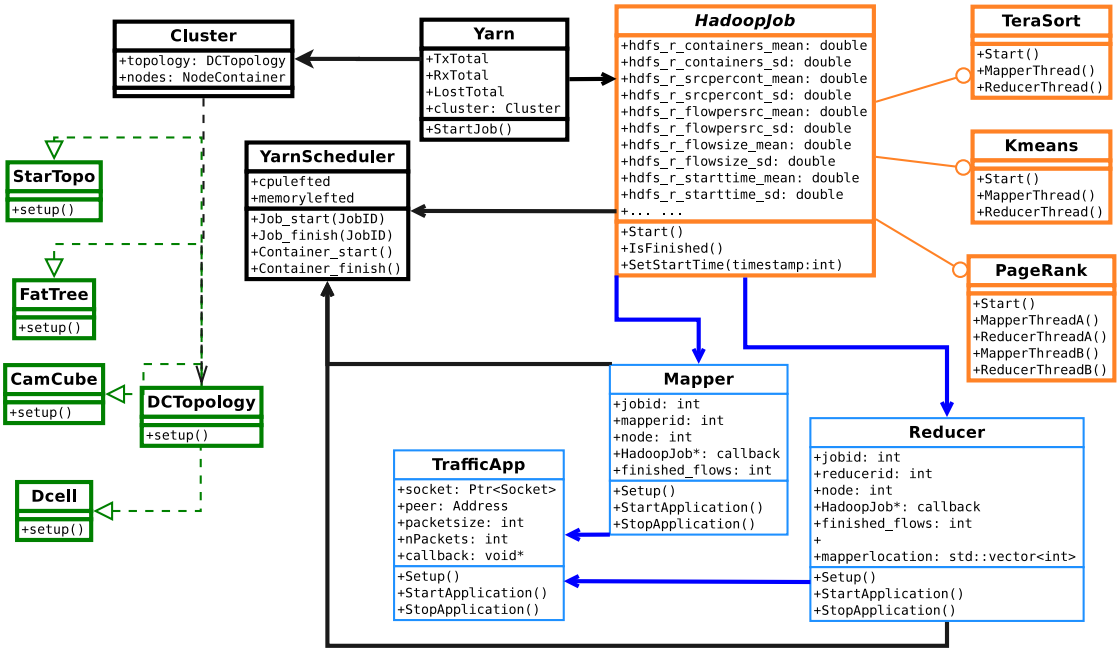

Figure 7.2: Classes implemented in Keddah, including cluster (green), Yarn (black), Hadoop jobs (orange) and containers (blue).

different colours. The arrows show the calling relations between classes. The rest of this section outlines their operation.

When running the simulation, users can choose from different cluster topologies and Hadoop jobs (just as Hadoop works in reality). Once given a certain topology and job, Keddah starts by building the cluster with a set of nodes interconnected by the stipulated topology (within ns3). Following this, then jobs are also created, which involves request placement on each node based on its available CPU and memory resources. Note that the placement is modelled as a container (*i.e.,* one physical node in the simulation can host multiple containers running a mapper or a reducer each container). Once the resource is allocated by the simulated YARN class, the containers are executed and then begin to generate traffic.

7.3.1.2 Implementing the cluster

When the simulation starts, it is first necessary to simulate the underlying cluster (data centre) topology and attach the "physical" nodes at the appropriate locations. As Keddah is built within ns3, it is not necessary to implement most functionality involved in this, *e.g.,* TCP/IP. Here, we only detail the ns3 components used to construct Keddah. The *Cluster* contains both the nodes (as a *ns3::NodeContainer*) and the *DCTopology* which defines how the nodes are connected. *DCTopology* is a parent class that must be extended by specific classes for each type of topology. As can be seen in Figure 7.2, we have implemented a number of example topologies already: *Star*, *FatTree*, *CamCube* and *DCell*. Other topologies can be implemented by following the same procedure. Each DCTopology class is expected to implement its own topology within the *setup()* method. For example, *FatTree* class defines the nodes, their IP addresses and their interconnections via its setup() method.

7.3.1.3 Implementing Yarn Resource Allocation

Yarn is the component that manages the Hadoop cluster, *i.e.,* the allocation of containers to nodes. We separate Yarn's responsibilities into two classes, shown in Figure 7.2: *Yarn* schedules the job submissions and *YarnScheduler* handles the resource allocation for scheduled jobs. The *ResourceScheduler* is the equivalent of the Resource Manager in Hadoop, it assigns the CPU and memory resources to the container when requested, *i.e.,* it allocates mapper and reducer containers to nodes within the simulator. CPU and memory are modelled for each node as a series of units; each node has a certain number of units, which are depleted by the placement of a container on the node. The *ResourceScheduler*, therefore, works as a queue that allocates containers (mappers and reducers) onto nodes that have sufficient resources. Though resource allocation is implemented via various schemes in Hadoop (including FIFO, fair and capacity), we have implemented a FIFO queue as a proof of concept. We have left the interface available

for other schemes to be added. Thus, when jobs are submitted to Yarn, it deals with them in a FIFO queue, allocating their mapper and reducer containers to specific nodes in the simulator one-by-one.

7.3.1.4 Implementing Jobs

The last part is executing the Hadoop jobs, where the traffic is generated. As an abstract class, *HadoopJob* declares the core attributes and methods needed within jobs. *HadoopJob* is then extended by any specific job class. This is because each job has their own set of distribution parameters according to Table 6-C. Thus, anybody can write a new job by extending this class.

7.3.1.5 Implementing Mappers and Reducers

Finally, we follow the ns3 design principle and implement Mapper and Reducer containers each as a *ns3::Application*. Each container is placed on a node within the simulator, and consumes a certain number of resource "units"; this is used to limit the number of containers per node. In other words, Yarn will only execute jobs and push the containers onto nodes once there are sufficient (modelled) resources to handle the complete job (*i.e.,* all containers). Each container is also equipped with a callback function so that the main job thread is notified when the container finishes; this imitates the resource allocation process of Hadoop in reality. Overall, by implementing the mapper and reducer containers as a *ns3::Application* we can leverage ns3 to have an independent running process for each container so that they can be simulated in parallel. To avoid repeating code within both the mapper and the reducer classes, we also implement a third class: *TrafficApp*. This is responsible for "orchestrating" all flows generating within the simulator. It reads in the Keddah file and uses its parameters to generate a timeline of flows to be initiated between the containers (*cf.* Section 7.3.2).

7.3.2 Simulating Single Jobs

Collectively, the above classes are used to generate traffic within the simulator. Because each job operates in a different fashion, the start method in each job implementation is customised. For example, the TeraSort starts the mapper threads and reducer threads only once, whilst the PageRank has two calculation stages, and each stage requires a new set of mappers and reducers. This highlights the need for different jobs to be implemented separately. Keddah currently has implementations for TeraSort and Kmeans; it is possible for others to implement their own jobs.

Figure 7.3 shows the traffic generation process in each job, including the job thread (black), mapper thread (green) and reducer thread (yellow). When the job starts in the simulator, the main thread first gets the number of mappers (from the Keddah file's five-tuple model). Note that the Keddah file might contain multiple five-tuple models, each for a different cluster configuration; hence, when running simulations, it is necessary to select the preferred configuration. It then starts the individual mapper threads. In the mapper threads, the process asks the *ResourceScheduler* for the required resource (*i.e.,* a working node to host itself). Once acquired, the node decrements the appropriate resource unit count from itself (using the *ResourceScheduler* and initiates the container. Note, CPU and memory resources are modelled (on each node) as possessing a certain number of "units", which can be stipulated by the experimenter (*e.g.,* 1GB of memory). Each container consumes a certain number of these units; by default, each container consumes 2xCores and 1GB of memory.

Once the mapper containers have been started, it follows the procedure below to

generate the traffic flows: Algorithm 1:

```
Data: the 5 Mapper traffic distributions
Result: generate the Mapper traffic
load number of sources (NS) from the Keddah file;
ns=GenerateRandomNumber.NormalDistribution(NS);
while ns do
    load number of flows per source (NFPS) from the Keddah file;
    nfps=GenerateRandomNumber.NormalDistribution(NFPS);
    SourceNode=GetRandomNodeFromCluster(); while nfps do
        load flow size (FS) and flow start time (FST) from the Keddah file;
        if FS.distribution==Binary then
            fs=GenerateRandomNumber.BinaryDistribution(FS);
        else
            fs=GenerateRandomNumber.WeibullDistribution(FS);
        end
        fst=GenerateRandomNumber.NormalDistribution(FST);
        TrafficApp(SourceNode,this,fs,fst);
    end
end
```

Algorithm 1: Generate the traffic of Mapper container.

And inside each flow, the traffic is generated using the procedure illustrated: Algorithm 2:

After all the mapper containers have been started, the job thread keeps a track of running/finished mappers. When the number of completed mappers exceeds the threshold (set by mapreduce.job.reduce.slowstart.completedmaps value in the Keddah source file), the job thread starts the reducers, which the total number of reducers yields to the distribution parameter in the Keddah file. Similar to the mapper, the reducer threads start with requesting resources, then generate traffic by starting the TCP application with the other four parameters from the 5-tuples in the Keddah file: number of sources, number

```
Data: SourceNode (sn),DestinationNode (ds),FlowSize (fs),FlowStartTime(fst)
Result: generate the Mapper traffic
wait(fst-currentTime);
sentSize=0;
while sentSize<=fs do
    packet(1000);
    SourcePort=random(65535);
    if Mapper flow then
        ns3.socket.send(sn,dn,SourcePort,50010,packet);
    else
        ns3.socket.send(sn,dn,13562,SourcePort,packet);
    end
    sentSize+=1000;
end
```

Algorithm 2: The TrafficApp class, generate one traffic flow.

of flow, flow size and start time. Finally, the HDFS write traffic is generated by the end of each reducer thread. And resources assigned to the container at the node is returned to the ResourceScheduler for new assignments.

7.3.3 Simulating Multiple Jobs

Keddah can also support running multiple jobs as a workload (within one simulation). This is typical in clusters, where multiple datasets will be processed in parallel across several jobs. However, simulating multiple nodes and jobs in parallel has always been challenging due to the difference between parallel processing with the discrete event simulations [244, 245]. We have explicitly built support for running multiple jobs with Keddah.

Simulating the traffic from multiple jobs in Keddah is similar to a single job: we generate the traffic from the application level so that that network layer behaviour can be exposed for study. We first tried to integrate OpenMP with ns3 to run jobs in parallel, however, OpenMP complicates the code implementation by specifying the behaviour for each computer core. We then tried pthread; however, the use of pthread in ns3 requires the real time simulator implementation and the simulation of Hadoop traffic is often

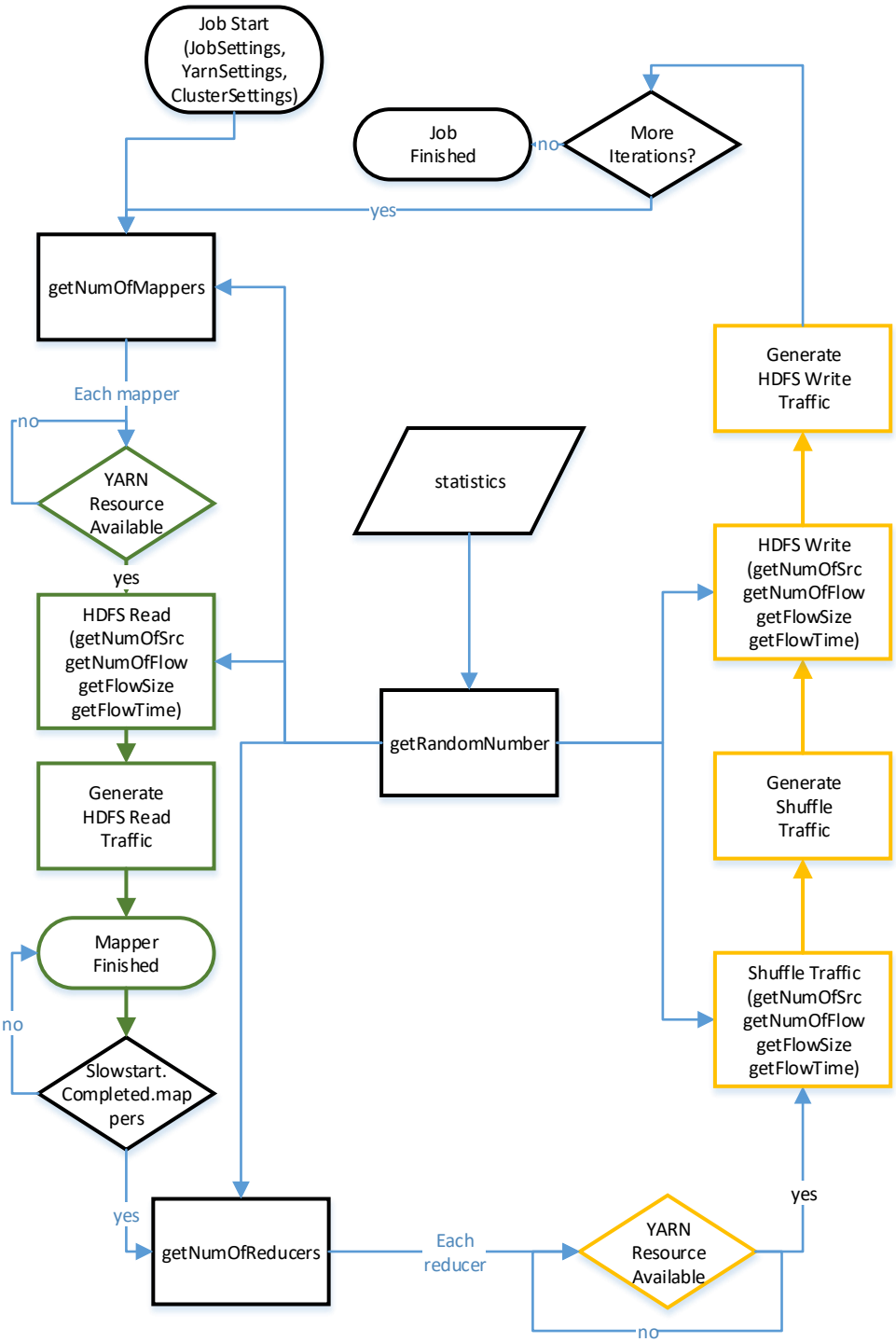

Figure 7.3: The flow chart for generating Hadoop traffic in ns3.

too heavy to be running in real time. So finally, we come up with a solution of each job implementation being expected to extend (inherits) the ns3 *Application* class. Upon executing the simulation, each job will enter a queue by an exponentially distributed interval (extracted from the real trace in SWIM [236]), and the job is only executed once sufficient resources (*i.e.,* simulated CPU and memory on the nodes) are available. At this point, the next job within the queue will begin execution. This reflects the process by which jobs are queued in a real Hadoop schedule.

Researchers can stipulate their own arrival rate for the multiple jobs. However, for convenience, we also make available the arrival rates based on the SWIM [243] dataset (which contains multiple jobs). To support multiple jobs, the ResourceScheduler we have implemented (which deals with resource management) allows multiple containers from multiple jobs to compete for the nodes within the simulator. Yarn will allocate containers to nodes one-by-one from each job in a round-robin fashion based on the resources available. Each job, therefore, executes as a single job, generating its own traffic within the simulator. We note that Keddah has an API allowing developers to implement any other scheduling mechanism they desire. The traffic is multiplexed within the IP-layer implementation of ns3.

7.4 Validation

By design, each of the individual empirical models (e.g., time distribution, amount of traffic) closely reflects the original traffic. However, it is important to validate that the combination of these attributes to replay the flow-level traffic is also accurate. To achieve this, we compare the original traffic against that replayed by Keddah. Ideally, these two sets should be statistically similar across the duration of the job execution.

7.4.1 By Correlation

We first compare the traffic of a single job by calculating the correlation between the simulated and the real traffic. The traffic is generated over a period of time and we want to include the time in the comparison as well. Thus, we first calculate the accumulated size of traffic transmitted over time for both the simulated and real trace and then calculate the correlation in between. Figure 7.4 shows the fraction of overall traffic generated per second across both our real testbed traces and our replayed traces (for TeraSort 2GB). We observe that, visually, the trends are similar, suggesting that the replayed traffic is similar.

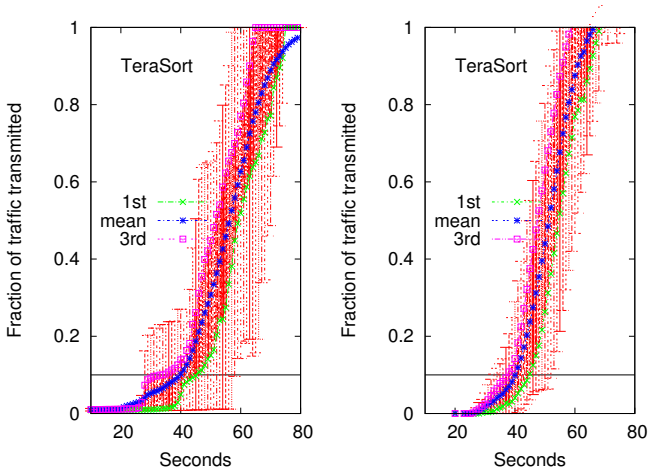

Figure 7.4: Cumulative traffic generated per second in (i) real physical testbed traces; and (ii) simulated replayed traces. The horizontal line demarcates HDFS Read from Shuffle traffic as shown in Figure 6.3.

To quantify the similarity, we calculate the correlation between the testbed traces and the simulated traffic. We calculate the correlations across the 1st quartile, mean and 3rd quartile of the accumulated percentage of traffic delivery shown in Figure 7.4. Table 7-B presents the correlations for several job types. We observe that the correlations are generally high, confirming that the 5 attributes, indeed, allow the traffic to be replayed.

Table 7-B: Correlation of cumulative traffic generated between simulation with real trace in terms of 1st quartile, mean and 3rd quartile.

Job	1st quartile	mean	3rd quartile
TeraSort 1GB	0.91	0.94	0.93
TeraSort 2GB	0.59	0.98	0.97
TeraSort 3GB	0.96	0.92	0.81
kmeans 10x5m	0.89	0.92	0.87
kmeans 10x8m	0.65	0.78	0.79

7.4.2 By Mean Absolute Percentage Error

To further validate the accuracy of Keddah, we run a set of TeraSort jobs with a fixed interval of 20 seconds between each job. We execute these in both the real testbed and the simulator environments and compare the traffic transmission. Figure 7.5 and Figure 7.6 show the comparison of the real and the simulation workload traffic using 3 and 5 TeraSort 3GB jobs. We can see that the simulated workload shows a very similar trend across time.

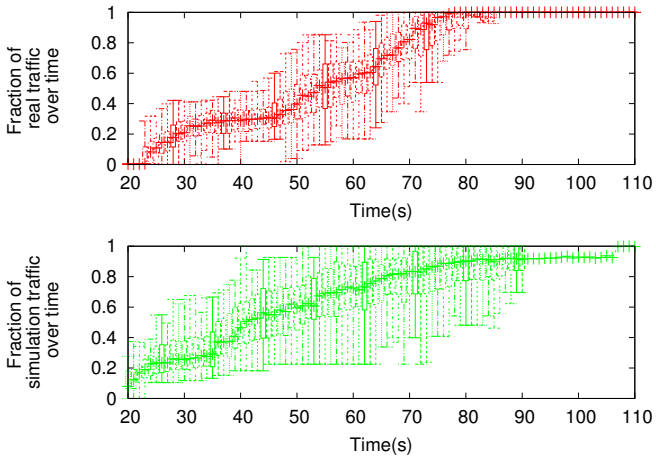

Figure 7.5: Comparison of traffic workload in reality and simulation when executing 3 TeraSort 3GB jobs.

To quantify the simulation accuracy, we introduce the mean absolute percentage error (MAPE) [246] to measure the distance between two transmissions (measured real *vs.* simulated):

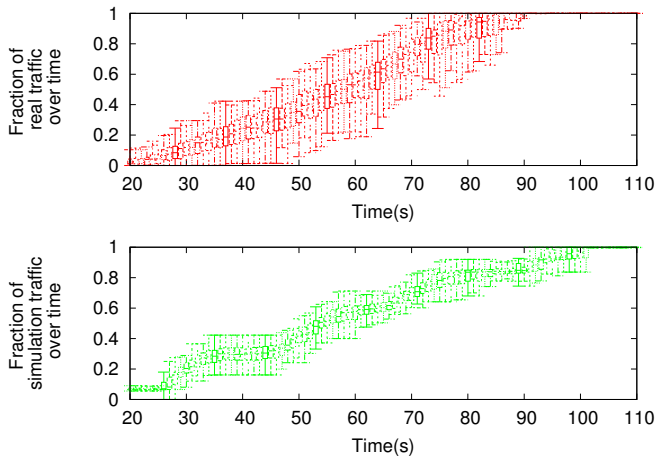

Figure 7.6: Comparison of traffic workload in reality and simulation when executing 5 TeraSort 3GB jobs.

Table 7-C: MAPE of simulated traffic transmission for workload consists of multiple jobs.

	Minimum	1st quartile	Median	3rd quartile	Maximum
TeraSort 3GB x 3	0.84	0.55	0.10	1.43	0.98
TeraSort 3GB x 5	0.49	0.26	0.09	0.97	0.56
Kmeans 100k*50 x 5	0.65	0.45	0.10	1.2	0.61
(TeraSort + Kmeans) x 5	0.55	0.7	0.95	0.98	0.60

$$MAPE = \sum_{t=start}^{end} \frac{|r_t - s_t|}{r_t} \tag{7.1}$$

Where t is the time, r_t is the fraction of traffic transmitted at time k over the real testbed, and s_t is the fraction of traffic transmitted at time t in the simulation. Due to the randomness in traffic, for each r_t and s_t, we use minimum value, 1st quartile, median, 3rd quartile and maximum values according to statistics and compare the MAPE respectively. Table 7-C shows the MAPE value calculated for TeraSort with different parameters. To gain a better confidence, apart from TeraSort, we also implemented Kmeans in Keddah and listed the accuracy in Table 7-C too. The distance between the simulated traffic with and real one is below 50% for most of the results, and below 20% in terms of median value. Thus we consider the difference is small enough to generate a valid Hadoop traffic from the simulation process.

Table 7-D: Experiment traffic model parameters.

Name	Decscription	Value
n	cluster size	$\begin{cases} FatTree: & 64 \\ CamCube: & 64 \\ DCell: & 42 \end{cases}$
b	mapred.max.split.size	128MB
s	TeraSort file size	3GB
clusters	Number of kmeans clusters	50
samples	Number of kmeans samples	50k

7.5 Use Case Demonstration

The goal behind Keddah is to allow researchers to evaluate network innovations under realistic application-layer Hadoop traffic. These include innovations such as novel data centre topologies, queuing mechanisms, routing protocols or congestion control algorithms. In this section, we demonstrate how Keddah can be used in profiling different network topologies and TCP protocol flavours, so that others can use this tool for their own purpose as well.

7.5.1 Experimental Setup

The workload we generate consists of an equal number of TeraSort (3G dataset) and kmeans (50 cluster and 50k samples) jobs, launched in a random sequence. The job parameters can be tuned for better flexibility, but we use a fixed job setup for testing purposes. We perform sensitivity analysis across the load by selecting different parameters for the job arrival process, modelled using an exponential distribution, as we found job arriving interval extracted from the real trace in SWIM [236] are exponential distributed.

7.5.2 Evaluating Network Topologies

A number of novel data centre topologies have been proposed in recent years, *e.g.,* FatTree [171], Dcell [177] and Camcube [176]. Thus, researchers developing such topologies

are required to evaluate the performance of applications (like Hadoop) operating over them. Previously, people have studied data centre topologies to see how the network changes as the data centre scales up [247]. However, doing so requires flexible environments (*e.g.,* simulations) that can easily vary these scale parameters. Keddah offers a powerful tool for such experiments.

Topology design involves a number of considerations, including performance, reliability, scalability, security and cost. Amongst which, performance is usually evaluated by the throughput, packet loss and latency (under various traffic loads) [16, 171, 182, 248]. However, for tractability, only a few types of traffic are typically used in real setups [249]. Thus, simulation methodologies are often used [128, 155, 156, 250] to investigated the network performance. Even so, the traffic is still so important because the performance of certain topologies may vary across different workloads and application patterns [128, 249].

To highlight the efficacy of Keddah, we next present experiments exploring the performance of various data centre topologies with Hadoop traffic, including FatTree [171], Dcell [177] and Camcube [176]. These three are very typical data center network topologies that are representatives of clos, star and torus topologies. The specific instances we use in the experiments are FatTree (8x8), CamCube (4) and DCell (2,2) which contains 64, 64, and 42 nodes respectively. Of course, Keddah could operate under any configurations the experimenter wishes.

We first measure the average throughput achieved among every nodes in the cluster by run the workload with various job submission interval. With the lower job submission interval, the traffic load will be higher. To get a precise result, we run each experiment 100 times and present the results from each run in a box plot. Figure 7.7 shows the throughput measured from all the three topologies under different workloads. We can see that, from the throughput point of view, topology is a determinate factor rather than workload intensity: DCell can achieve 10x high throughput than CamCube and nearly 50x more than FatTree. Though the difference is getting smaller with low-intensity work-

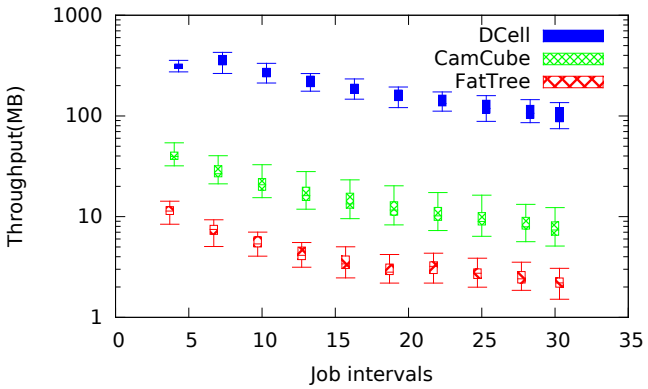

Figure 7.7: Comparison of the average throughput of three topologies in different workloads. The same result can be found from previous work

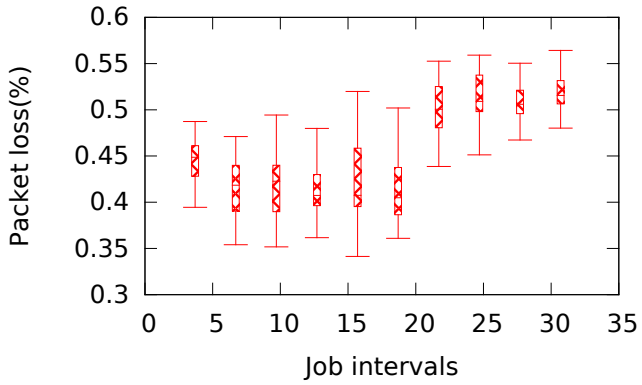

Figure 7.8: Comparison of the packet loss in the FatTree topology.

loads, DCell still holds more advantages than CamCube and FatTree. It is worth noting that the throughput results shown in our simulation match the results of works [128, 249] in terms of the all-to-all or randomly selected traffic matrices.

Moreover, we also evaluate the packet loss over various workloads. Again, we run each experiment 100 times to show the results in a box plot. Fig. 7.8 shows the percentage of packet loss compared to the total traffic delivered in FatTree. Strangely, although the figure is floating within a certain range, the percentage of packet loss is increased when the workload intensity decreases. We failed to found any previous study on this as a reference.

7.5.3 TCP vs DCTCP

Our next experiment explores the performance benefits of Data Center TCP (DCTCP) [196] — a flavour of TCP specifically designed for use in data centres. Unlike the traffic patterns that DCTCP was originally evaluated with, we have found that Hadoop generates many short flows with high burstiness. We use the DCTCP implementation in ns3 [2], and RED queues [185] with a marking threshold at 100 and 40 packets, respectively. The workload consists of TeraSort and Kmeans jobs, running on the FatTree and CamCube topologies.

We compare the performance achieved of TCP NewReno [251] against DCTCP with various topologies and workload intensities. Figure 7.9 and Figure 7.10 show the throughput and packet loss measured for FatTree and CamCube under various workloads. Again, as each experiment runs over 100 times, the measure is shown as a range.

We found that DCTCP, indeed, attains improved throughput, however, depends on the type of topology. We measure the average network throughput among all nodes, against various workload traffic load represented by the interval in between jobs submitted. Again, each experiment is run a set of times thus the throughput measured in shown by a box plot. As shown in Figure 7.9, DCTCP increased the throughput beyond TCP NewReno by between 1 and 14% (the best case was under a heavy workload, where TCP NewReno performed poorly). However, when it comes to CamCube, as shown in Figure 7.10, the throughput of normal TCP can be higher than the DCTCP, especially when the workload is high intensity. Similar results can be found in [252] showing regular TCP can achieve higher throughput when the traffic load is low.

In terms of packet loss, we did the same measurement and plot the overall packet loss among all node in Figure 7.11, against various traffic load. As expected, DCTCP achieves a lower packet loss rate across all workloads around 0.25%, where for TCP NewReno is around 0.45%. We also note that the packet loss is particular lower during

[2] https://github.com/i-maravic/ns-3

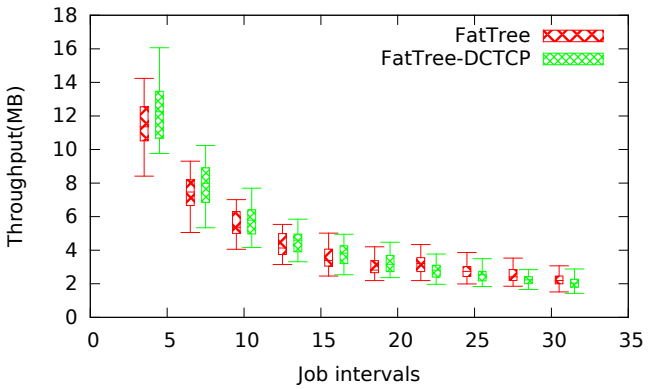

Figure 7.9: Comparison of the throughput of TCP NewReno with DCTCP with FatTree under different workload, higher the better.

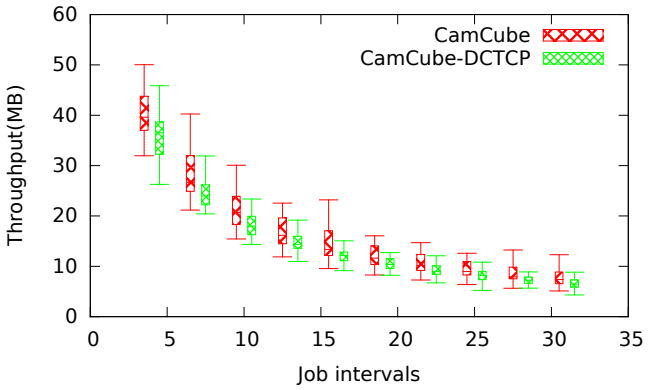

Figure 7.10: Comparison of the throughput of TCP NewReno with DCTCP with CamCube under different workload, higher the better.

heavy workloads. If we correlated the packet loss with the throughput, we can see that as the packet loss is reduced by 0.1% to 0.2%, the throughput of DCTCP is increased by 4.2% to 6.4% in the heavy loads shown in Figure. 7.9.

7.5.4 Conclusion on Keddah Use Case

In conclusion, we have shown two examples of how researchers might use Keddah to profile network innovations over Hadoop cluster, *i.e.,* novel topologies, and protocols.

Previously, due to the limited experimental methods and traffic generators, people could not investigate how Hadoop might interact with the network via simulations. How-

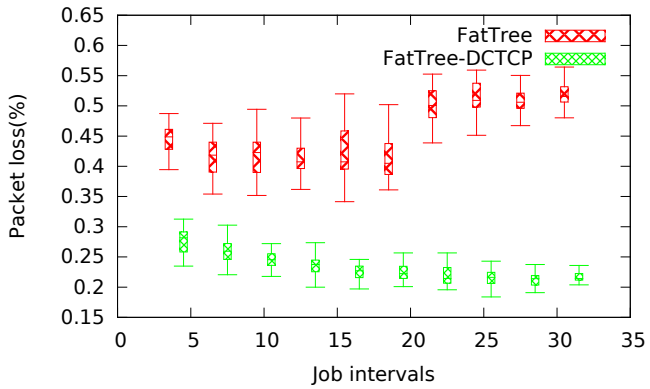

Figure 7.11: Comparison of the packet loss (in percentage) of TCP NewReno with DCTCP with FatTree under different workload, lower the better.

ever, as the two examples have highlighted, Keddah allows researchers to explore their network-level innovations with realistic Hadoop traffic running over them. By using Keddah, we revealed the performance of various network topologies and protocols. We made a series of findings: we found DCell outperforms FatTree and CamCube which can be used as a guide when choosing the network topology for distributed system cluster. We also evaluated the performance of DCTCP, found that DCTCP's performance is dependent on the underlying topology. That said the protocol performance could not be determined without taking the underlay network and application into consideration. We emphasise that this is not meant as an exhaustive list of experiments that Keddah supports — simply two exemplars of what Keddah *can* support.

7.6 Summary

We have presented a methodology and a tool for capturing the traffic of Hadoop jobs, in order to generate traffic workloads for network simulations. Our empirical measurements show how different jobs have unique traffic patterns. Keddah uses the captured information to generate traffic workloads parametrised to the main characteristics of the job and the dataset.

Keddah greatly simplifies the simulation of Big Data workloads, which we believe can help with the initial evaluation of data centre network research initiatives. This chapter has shown how the integration with a network simulator enables researchers to evaluate very different network components such as data centre topologies and congestion control protocol variants with respect to the workload of a distributed data processing job. This tool fulfilled our goal of enabling the exploration of various networking techniques on the distributed processing applications, so that networking performance of distributed processing applications can be further studied.

Part V

Conclusion

Chapter 8

Conclusion and Future Work

8.1 Summary

Networking has developed over many years, and people have proposed many techniques to help improve network performance for delay, throughput, reliability, etc. However, network performance is still so fragile that any alteration that impacts the traffic can hugely change the result. Also, the diversity of applications makes networks even more difficult to manage. Thus, any evidence we can find regarding the traffic generated by popular applications can be useful to guide network operations.

However, with the fast evolution of network applications, most previous work on profiling applications is no longer enough. For example, people have done a lot of work on profiling large-scale Internet applications such as video-on-demand (VOD) and user-generated-content (UGC), but little work has examined user-generated live streaming. Similar to data center applications, many works have been done on characterising data center traffic such as Web applications. However very few of them look at distributed processing applications like Hadoop. This lack of understanding has prevented people from proposing network side innovations to better handle of the traffic. Instead, the network can only work in a best-effort manner as we do not know what support those

applications need from the network.

The first step towards the understanding of application traffic patterns is to measure and characterise the application traffic. In this thesis, two popular distributed applications have been investigated, Twitch and Hadoop, representative of Internet-wide and datacentre-scale respectively:

- Twitch is a recently-emerged single application; it gathers 600k users online on average and has become the 4th peak US Internet traffic generator.
- In the data centre, distributed applications built with the Hadoop framework are widely deployed and developed, these have not been sufficiently investigated from the network side.

As we can't fully acquire the traffic of Twitch from the Internet, we approached discovering the popularity behind this large-scale online video streaming platform by using public web crawling. We started by investigating the content, as the content is what has driven the popularity in non-live UGC platforms. After correlating the metadata of content in terms of genre, release date and rating, we found that the channel popularity is barely related to the content (the games being broadcast). Thus, we continued to profile the streamers: the streamers who are professional players, the streamers who are trying to finish the games with minimum time (*speedrun*). One of the bot streamers attracted millions of people watching, and many other contributors stream in their own unique ways. They can even carry with them all the viewers of that game when they switch to other games. With this understanding of popularity, we continued to investigate the content delivery techniques of Twitch. We used open HTTP proxies to perform a large-scale measurement study of Twitch's infrastructure and revealed Twitch's global infrastructure deployment in North America, Europe and Asia. We found that Twitch uses a cost-effective streaming placement strategy which first places the stream in North America and then distributes it based on the stream popularity. Finally, we found that Twitch has weak client redirection in the Asia area which could be caused by its low

levels of Internet peering in Asia. This finding underlined the importance of networking for large-scale services like Twitch.

For Hadoop, we started with profiling the network traffic behaviours, investigating traffic generated for various jobs, such as different clusters, cluster settings and job settings. To quantify such traffic, we proposed a traffic matrix formed of a set of distributions, so that traffic generated under different scenarios could be compared, reproduced and even extrapolated. Furthermore, we implemented this traffic reproduction function in a state-of-the-art network simulator named ns3, so that people can evaluate different network components handling Hadoop traffic. Finally, a few examples of using such simulator are shown, including comparing the performance of different DCN topologies and transport layer protocols.

At the end of this research, we have acquired an in-depth understanding of the traffic behaviour of two distributed applications, investigated how the traffic is handled at the moment and how the traffic delivery can be evaluated. This work is valuable in providing insightful findings for application behaviour related study, and also enabling the evaluation and prediction of network performance.

8.2 Contributions

This thesis provides a base observation of traffic behaviour of distributed applications as different scales. Through data-driven studies including measurement, modelling and simulation, a clear understanding of the network traffic generated by both applications is presented, and from these application behaviours, the networking architecture and performance have been explored. The specific contributions are listed here:

1. **We revealed the traffic patterns of two popular distributed applications.**

   This thesis carefully surveyed a number of distributed applications and found a gap between the deployment of distributed applications and the understanding of

networking for those applications. Although, it's essential for the distributed applications to perform, the networking impact of user-generated online video streaming and distributed data processing application is poorly understood. Thus, this thesis found some of the most popular and representative applications in each area, namely Twitch and Hadoop, and used them to enrich the understanding of networking impacts. We have characterised the behaviour of both applications, either by actively and passively probe the system status, or establish a test environment to generate the traffic by ourselves. The behaviour we collected is the first of its kind, and can be used as a reference for measurement works in the future.

2. **We revealed that streamers are the key factor of popularity in Twitch.**

   A major contribution of this thesis is identifying the key factor behind the popularity of user-generated online video streaming: streamers. We used public APIs to collect data from the most popular user-generated online video streaming platform, Twitch. Like other studies on user-generated content platforms, we first investigated the popularity of Twitch against the contents. As a video game streaming platform, we used game metadata from an external source to estimate the Twitch channel popularity. However, we found that unlike traditional VOD platforms, the popular contents are usually newly generated. Though newly released games in Twitch can draw little attention, the top few games in Twitch remain popular for years. Those top games are usually broadcast during tournaments, which can produce peak viewer figures. This phenomenon led us to investigate the channels behind the games, and we found the popularity is more correlated with the channel, for example, whatever game the channel decides to play, it always draws a lot of viewers. Players/channels can draw all the viewers from one game to another, and their popularity can even go beyond Twitch to other social platforms.

3. **We discovered Twitch uses a single global-scale CDN without caching to deliver user-generated live video streaming.**

The infrastructure footprint of the largest user-generated online video streaming platform has been revealed in this thesis and how the infrastructure is utilized has been justified. To reduce the latency of content delivery and to enable real-time interaction during the streaming, Twitch only uses a dozen POPs around four continents: NA, EU, AS and OC. This is unlike other VOD services that often deploy content mirroring in ISP caches. To reduce the hosting overload, the stream is always hosted in NA first and spread across other continent servers based on the channel popularity in the region. Clients are redirected to one of the Twitch POPs based on their Internet routing. However, the performance of this redirection strategy is limited by Twitch's peering in the Asian area, which leads to many Asian clients being served by NA servers.

4. **We proposed a traffic model to capture Hadoop traffic, so that Hadoop traffic can be compared and reproduced.**

   This thesis is the first one to characterise and model the traffic generated by the Hadoop distributed processing framework. Though networking is essential to perform any data processing task, little work has been done on the networking aspect of Hadoop. In this thesis, we first characterised the Hadoop traffic under various cluster settings, jobs and job settings. Based on the characterisation, we further proposed a traffic model that takes the Hadoop behaviour into consideration and can capture and describe the Hadoop traffic with fine granularity.

5. **We developed Keddah, a simulation tool for profiling the networking performance with Hadoop traffic.**

   To enable the study of networking support for Hadoop without extra infrastructure cost, and reduce the cost of modifying topologies and protocols, a tool named Keddah was built, that can work along with the popular network simulator ns3 to explore various network components for Hadoop. By using this tool, people can generate Hadoop job traffic workload while evaluating the performance of differ-

ent networking technologies, to explore the best networking strategy for Hadoop traffic. We have shown a few examples of Hadoop traffic performance under various data centre networking topologies and transport layer protocols using Keddah. This work can be easily extended using a similar approach to explore the network performance of distributed processing applications.

8.3 Limitations

Two popular distributed applications named Twitch and Hadoop have been profiled in this thesis. However, there are some limitations due to the data sources and resources.

On the Twitch side, we have profiled the content popularity and content delivery to understand how the traffic is generated and delivered. To fully reveal how Twitch delivers content across the globe, we took advantage of Internet proxies and leveraged the locations provided. We profiled how Twitch handles traffic delivery in different regions. However, due to the traffic visibility on the Internet, we can not fully see the traffic end to end. For example, we used proxies to simulate the client: we found the proxy server location A and probed the Twitch broadcast server's location B. However, we cannot see how the content is transmitted to B from an internal Twitch source. Lack of this information makes it difficult to explain some of the behaviours observed, such as why servers in Asia are less likely to be selected. We found explanations from external sources such as BGP peering, but hard evidence is still missing to prove that. If traffic from the central Twitch servers could be found, then we could investigate different aspects such as cost and performance to understand the stream delivery and redirection policy.

For Hadoop, we can successfully generate Hadoop traffic for research studies by reproducing the empirical traces captured on our 16-node testbed. As a generic data processing platform, Hadoop can support various jobs based on the MapReduce platform. In our work, we profiled popular jobs commonly used in benchmarking such as TeraSort, Kmeans, PageRank and WordCount. Those jobs are also utilised by the research com-

munity in profiling Hadoop. However, jobs from production environments should be included to reveal the full spectrum of Hadoop traffic. With limited access to industrial environments and data, currently we do not have traffic samples from the production environment, and this certainly reduces the reliability of our network simulation tool. From the implementation point of view, we have not generalized the traffic generation of different jobs as each job needs to be implemented separately. Other work we can do to address the generality is to extend the traffic model to other new distributed processing frameworks, such as Apache Tez [140]. We hope our open-source network simulation tool will encourage its use by the industry to publicise their traffic statistics so that the community can be benefit from a diversity of traffic profiles.

8.4 Future Work

Apart from the work shown in this thesis, there are still many further directions that can be explored based on current progress. For example, we have successfully profiled the network behaviour of Hadoop and proposed a method to evaluate the performance of network components based on Hadoop traffic. Thus, we can estimate the network performance of a cluster before it is established. However, the network is not included alongside CPU and memory in the resource management of Hadoop, despite the fact that the network can massively affect the distributed system performance. That said, one of the future work streams of the Hadoop team is to include the network (*e.g.,* bandwidth, queue, *etc.*) as a resource in the resource scheduling. With the network scheduler implemented, the tasks could be prioritised according to network conditions. One of the main challenges of this work is the scheduling algorithm, which could use the traffic characterisation results of this thesis as the foundation. By introducing the Hadoop traffic distributions, one can schedule the traffic flow using linear programming to optimise the overall link utilisation or packet latency.

Much work could also be carried out with developing a large-scale video streaming

platform Twitch. Based on the content popularity and current infrastructure we have already found, we can explore lots of server allocation algorithms such as allocating the servers based on predictions of the viewing demand to use the servers more efficiently.

8.5 Concluding Remarks

This thesis has profiled two distributed applications operating at different scales. The distributed systems are heavily relying on the underlying network. However, their network behaviour is not well understood currently. This makes it difficult to find the right networking techniques to support these applications, not to say improving network performance. Thus, two popular yet poorly understood distributed applications have been studied in two parts of this thesis. The first part focused on a large-scale live video streaming platform. We first investigated the popularity to reveal the viewing behaviour. Based on the popularity, the worldwide content delivery infrastructure is studied to reveal the networking aspect of live streaming. The second part of this thesis focused on a data center scale distributed processing application. Similarly, we first characterised the application traffic. Based on the modelling result for the traffic, a distributed application traffic generation tool has been proposed. With such tool, people can profile various network components against distributed application traffic, enabling network performance explorations on distributed applications. With the distributed applications becoming more and more popular nowadays, the characterised traffic patterns and tools developed in this thesis will further help people improve the network performance for distributed applications.

References

[1] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design," in *Proceedings of the 11th USENIX conference on networked systems design and implementation. USENIX Association, Berkeley*, 2014, pp. 29–41.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[3] R. Buyya *et al.*, "High performance cluster computing: Architectures and systems (volume 1)," *Prentice Hall, Upper SaddleRiver, NJ, USA*, vol. 1, p. 999, 1999.

[4] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[5] J. Moore, J. Chase, K. Farkas, and P. Ranganathan, "Data center workload monitoring, analysis, and emulation," in *Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2005.

[6] D. Ersoz, M. S. Yousif, and C. R. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," in *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*. IEEE, 2007, pp. 59–59.

[7] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 253–266.

[8] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[9] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *Network, iEEE*, vol. 11, no. 6, pp. 10–23, 1997.

[10] K. Claffy, G. Miller, and K. Thompson, "The nature of the beast: Recent traffic

measurements from an internet backbone," in *Proceedings of INET*, vol. 98, 1998, pp. 21–24.

[11] N. Brownlee and K. Claffy, "Understanding internet traffic streams: dragonflies and tortoises," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 110–117, 2002.

[12] A. M. Odlyzko, "Internet traffic growth: Sources and implications," in *ITCom 2003*. International Society for Optics and Photonics, 2003, pp. 1–15.

[13] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 90–102.

[14] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 123–137.

[15] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM, 2011, p. 25.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[17] "Google Search Engine," www.google.com.

[18] "facebook," facebook.com.

[19] "Twitter," https://twitter.com.

[20] "YouTube," www.youtube.com.

[21] "Netflix - Watch TV Shows Online, Watch Movies Online," www.netflix.com.

[22] R. Li, G. Gao, W. Xiao, and Z. Xu, "Measurement study on pplive based on channel popularity," in *Communication Networks and Services Research Conference (CNSR), 2011 Ninth Annual*. IEEE, 2011, pp. 18–25.

[23] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, every-

body tubes: analyzing the world's largest user generated content video system," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 1–14.

[24] ——, "Analyzing the video popularity characteristics of large-scale user generated content systems," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 5, pp. 1357–1370, 2009.

[25] S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno, "Analysis of pplive through active and passive measurements," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–7.

[26] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 248–257.

[27] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 1620–1628.

[28] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '12)*. ACM, 2012, pp. 359–370.

[29] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *Proceedings of the 17th ACM international conference on Multimedia*. ACM, 2009, pp. 25–34.

[30] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "Youtube traffic dynamics and its interplay with a tier-1 isp: an isp perspective," in *Proceedings of the 2010 ACM Conference on Internet Measurement (IMC '10)*. ACM, 2010, pp. 431–443.

[31] ——, "Where do you tube? uncovering youtube server selection strategy," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. IEEE, 2011, pp. 1–6.

[32] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z.-L. Zhang, "A tale of three cdns: An active measurement study of hulu and its cdns," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012, pp. 7–12.

[33] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM Conference on Internet Measurement (IMC '11)*. ACM, 2011, pp. 345–360.

[34] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn," *arXiv:1606.05519*, 2016.

[35] "Twitch ranked 4th in peak internet traffic," http://www.gamespot.com/.

[36] "Twitch," https://www.twitch.tv/.

[37] "YouTube Gaming," https://gaming.youtube.com/.

[38] "Periscope Mobile Streaming," https://www.periscope.tv/.

[39] "Essential facts about the computer and video game industry," http://www.theesa.com/facts/pdfs/ESA_EF_2014.pdf.

[40] "Worldwide game industry hits $91 billion in revenues in 2016," https://venturebeat.com/2016/12/21/worldwide-game-industry-hits-91-billion-in-revenues-in-2016-with-mobile-the-clear-leader/.

[41] "Statistics and facts about the video game industry," http://www.statista.com/topics/868/video-games.

[42] "Twitch: THE 2015 RETROSPECTIVE," https://www.twitch.tv/year/2015.

[43] K. Pires and G. Simon, "Youtube live and twitch: A tour of user-generated live streaming systems," in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys '15. New York, NY, USA: ACM, 2015, pp. 225–230.

[44] M. Siekkinen, E. Masala, and T. Kämäräinen, "A first look at quality of mobile live streaming experience: the case of periscope," in *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 2016, pp. 477–483.

[45] A. Jacobs, "The pathologies of big data," *Communications of the ACM*, vol. 52,

no. 8, pp. 36–44, 2009.

[46] "Teradata - DBC/1012," http://www.computerhistory.org/collections/catalog/102713120.

[47] "HPCC SYSTEMS," https://hpccsystems.com.

[48] "LexisNexis Group Inc." https://www.lexisnexis.com.

[49] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[50] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[51] "World Hadoop Market - Opportunities and Forecasts, 2020," https://www.alliedmarketresearch.c hadoop-market.

[52] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The performance of mapreduce: An in-depth study," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 472–483, 2010.

[53] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda, "Can high-performance interconnects benefit hadoop distributed file system," in *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO*. Citeseer, 2010.

[54] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Proc. IPDPSW*. IEEE, 2010.

[55] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "Cohadoop: flexible data placement and its exploitation in hadoop," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, pp. 575–585, 2011.

[56] C. L. Abad, Y. Lu, and R. H. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. Ieee, 2011, pp. 159–168.

[57] W. Wang and L. Ying, "Data locality in mapreduce: A network perspective," *Performance Evaluation*, vol. 96, pp. 1–11, 2016.

[58] "YouTube statistics," www.youtube.com/yt/press/en-GB/statistics.html.

[59] "Number of netflix streaming subscribers," www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/.

[60] "Twitch dominated streaming in 2013," www.dailydot.com/esports/twitch-growth-esports-streaming-mlg-youtube-2013/.

[61] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: Youtube network traffic at a campus network-measurements and implications," *Computer Science Department Faculty Publication Series*, p. 177, 2008.

[62] Y. Zhu, G. Zheng, K.-K. Wong, S. Jin, and S. Lambotharan, "Performance analysis of cache-enabled millimeter wave small cell networks," *IEEE Transactions on Vehicular Technology*, 2018.

[63] Y. Zhu, G. Zheng, L. Wang, K.-K. Wong, and L. Zhao, "Content placement in cache-enabled sub-6 ghz and millimeter-wave multi-antenna dense small cell networks," *IEEE Transactions on Wireless Communications*, 2018.

[64] "How content delivery networks work," https://www.cdnetworks.com/en/news/how-content-delivery-networks-work/4258.

[65] L. Li, D. Alderson, W. Willinger, and J. Doyle, "A first-principles approach to understanding the internet's router-level topology," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 3–14.

[66] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 4, pp. 56–69, 2007.

[67] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology discovery in heterogeneous ip networks," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 2000, pp. 265–274.

[68] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering internet topology," *Unpublished manuscript*, 1998.

[69] B. Huffaker, D. Plummer, D. Moore, and K. Claffy, "Topology discovery by active probing," in *Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on.* IEEE, 2002, pp. 90–96.

[70] N. Spring, M. Dontcheva, M. Rodrig, and D. Wetherall, "How to resolve ip aliases,"

Technical report, Univ. of Washington, Tech. Rep., 2004.

[71] M. H. Gunes and K. Sarac, "Importance of ip alias resolution in sampling internet topologies," in *IEEE Global Internet Symposium, 2007*. IEEE, 2007, pp. 19–24.

[72] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker, "In search of path diversity in isp networks," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 313–318.

[73] M. Zhang, Y. Ruan, V. S. Pai, and J. Rexford, "How dns misnaming distorts internet topology mapping." in *USENIX Annual Technical Conference, General Track*, 2006, pp. 369–374.

[74] H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger, "Towards capturing representative as-level internet topologies," *Computer Networks*, vol. 44, no. 6, pp. 737–755, 2004.

[75] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transactions on networking*, vol. 9, no. 6, pp. 733–745, 2001.

[76] Z. M. Mao, D. Johnson, J. Rexford, J. Wang, and R. Katz, "Scalable and accurate identification of as-level forwarding paths," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2004, pp. 1605–1615.

[77] S. Zhou and R. J. Mondragón, "The rich-club phenomenon in the internet topology," *IEEE Communications Letters*, vol. 8, no. 3, pp. 180–182, 2004.

[78] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM computer communication review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[79] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in internet topologies," *ACM SIGCOMM computer communication review*, vol. 30, no. 2, pp. 18–28, 2000.

[80] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 2002, pp. 638–647.

[81] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1587–1596.

[82] A.-M. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, vol. 4, 2007.

[83] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting youtube: An active measurement study," in *INFOCOM, 2012 Proceedings IEEE.* IEEE, 2012, pp. 2521–2525.

[84] D. A. Ferguson and E. M. Perse, "The world wide web as a functional alternative to television," *Journal of Broadcasting & Electronic Media*, vol. 44, no. 2, pp. 155–174, 2000.

[85] B. Thielmann and M. Dowling, "Convergence and innovation strategy for service provision in emerging web-tv markets," *International Journal on Media Management*, vol. 1, no. 1, pp. 4–9, 1999.

[86] Y. Ninomiya, Y. Ohtsuka, Y. Izumi, S. Gohshi, and Y. Iwadate, "An hdtv broadcasting system utilizing a bandwidth compression technique-muse," *IEEE transactions on broadcasting*, no. 4, pp. 130–160, 1987.

[87] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h. 264/avc standard," *IEEE Transactions on circuits and systems for video technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[88] N. Liu, H. Cui, S.-H. G. Chan, Z. Chen, and Y. Zhuang, "Dissecting user behaviors for a simultaneous live and vod iptv system," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 10, no. 3, p. 23, 2014.

[89] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 2102–2111.

[90] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale p2p iptv system," vol. 9, no. 8, pp. 1672–1687, 2007.

[91] A. Horvath, M. Telek, D. Rossi, P. Veglia, D. Ciullo, M. A. Garcia, E. Leonardi, and M. Mellia, "Dissecting pplive, sopcast, tvants," *submitted to ACM Conext*, vol. 30, 2008.

[92] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 41–54.

[93] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, 2008.

[94] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4. ACM, 2006, pp. 333–344.

[95] Y. Chen, Y. Yu, W. Zhang, and J. Shen, "Analyzing user behavior history for constructing user profile," in *IT in Medicine and Education, 2008. ITME 2008. IEEE International Symposium on*. IEEE, 2008, pp. 343–348.

[96] B. Hu, M. Jamali, and M. Ester, "Learning the strength of the factors influencing user behavior in online social networks," in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. IEEE Computer Society, 2012, pp. 368–375.

[97] V. Gopalakrishnan, R. Jana, R. Knag, K. Ramakrishnan, D. F. Swayne, and V. A. Vaishampayan, "Characterizing interactive behavior in a large-scale operational iptv environment," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–5.

[98] B. Chang, L. Dai, Y. Cui, and Y. Xue, "On feasibility of p2p on-demand streaming via empirical vod user behavior analysis," in *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*. IEEE, 2008, pp. 7–11.

[99] D. Krishnappa, S. Khemmarat, L. Gao, and M. Zink, "On the feasibility of prefetching and caching for online tv services: a measurement study on hulu," in *Passive and Active Measurement*. Springer, 2011, pp. 72–80.

[100] A. Zeng, S. Gualdi, M. Medo, and Y.-C. Zhang, "Trend prediction in temporal bipartite networks: the case of movielens, netflix, and digg," *Advances in Complex Systems*, vol. 16, no. 04n05, p. 1350024, 2013.

[101] D. Karamshuk, N. Sastry, A. Secker, and J. Chandaria, "Isp-friendly peer-assisted on-demand streaming of long duration content in bbc iplayer," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 289–297.

[102] M. Zink, *Scalable video on demand: adaptive internet-based distribution*. John Wiley & Sons, 2013.

[103] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," 2007, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298309

[104] G. Tyson, Y. Elkhatib, N. Sastry, and S. Uhlig, "Demystifying porn 2.0: a look into a major adult video streaming website," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 417–426.

[105] S. Ali, A. Mathur, and H. Zhang, "Measurement of commercial peer-to-peer live video streaming," in *Proc. of Workshop in Recent Advances in Peer-to-Peer Streaming*, 2006, pp. 1–12.

[106] T. Silverston and O. Fourmaux, "Measuring p2p iptv systems," in *Proceedings of NOSSDAV*, vol. 7, 2007, p. 2.

[107] "Facebook Live," https://live.fb.com/.

[108] E. A. Vandewater, M.-s. Shim, and A. G. Caplovitz, "Linking obesity and activity level with children's television and video game use," *Journal of adolescence*, vol. 27, no. 1, pp. 71–85, 2004.

[109] D. A. Gentile, P. J. Lynch, J. R. Linder, and D. A. Walsh, "The effects of violent video game habits on adolescent hostility, aggressive behaviors, and school performance," *Journal of adolescence*, vol. 27, no. 1, pp. 5–22, 2004.

[110] K. Lucas and J. L. Sherry, "Sex differences in video game play: A communication-based explanation," *Communication research*, vol. 31, no. 5, pp. 499–523, 2004.

[111] L. M. Padilla-Walker, L. J. Nelson, J. S. Carroll, and A. C. Jensen, "More than a

just a game: video game and internet use during emerging adulthood," *Journal of youth and adolescence*, vol. 39, no. 2, pp. 103–113, 2010.

[112] S. Gallagher and S. H. Park, "Innovation and competition in standard-based industries: a historical analysis of the us home video game market," *IEEE transactions on engineering management*, vol. 49, no. 1, pp. 67–82, 2002.

[113] T. Chung, J. Han, D. Choi, T. T. Kwon, H. K. Kim, and Y. Choi, "Unveiling group characteristics in online social games: a socio-economic analysis," in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 889–900.

[114] J. Blackburn and H. Kwak, "Stfu noob!: predicting crowdsourced decisions on toxic behavior in online games," in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 877–888.

[115] R. Kowert and J. A. Oldmeadow, "Playing for social comfort: Online video game play as a social accommodator for the insecurely attached," *Computers in human behavior*, vol. 53, pp. 556–566, 2015.

[116] T. Smith, M. Obrist, and P. Wright, "Live-streaming changes the (video) game," in *Proceedings of the 11th european conference on Interactive TV and video*. ACM, 2013, pp. 131–138.

[117] D. Ramirez, J. Saucerman, and J. Dietmeier, "Twitch plays pokemon: A case study in big g games," 2014.

[118] K. Pires and G. Simon, "Youtube live and twitch: a tour of user-generated live streaming systems," in *Proceedings of the 6th ACM Multimedia Systems Conference*. ACM, 2015, pp. 225–230.

[119] F. Chen, C. Zhang, F. Wang, and J. Liu, "Crowdsourced live streaming over the cloud," *arXiv preprint arXiv:1502.06314*, 2015.

[120] K. Pires and G. Simon, "Dash in twitch: Adaptive bitrate streaming in live game streaming platforms," in *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. ACM, 2014, pp. 13–18.

[121] G. Nascimento, M. Ribeiro, L. Cerf, N. Cesario, M. Kaytoue, C. Raissi, T. Vasconcelos, and W. Meira, "Modeling and analyzing the video game live-streaming community," in *Web Congress (LA-WEB), 2014 9th Latin American*. IEEE, 2014,

pp. 1–9.

[122] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: A twitch. tv-based measurement study," *arXiv preprint arXiv:1502.04666*, 2015.

[123] M. Kaytoue, A. Silva, L. Cerf, W. Meira Jr, and C. Raïssi, "Watch me playing, i am a professional: a first study on video game live streaming," in *Proceedings of the 21st international conference companion on World Wide Web*. ACM, 2012, pp. 1181–1188.

[124] M. Claypool, D. Farrington, and N. Muesch, "Measurement-based analysis of the video characteristics of twitch. tv," in *Games Entertainment Media Conference (GEM), 2015 IEEE*. IEEE, 2015, pp. 1–4.

[125] R. Shea, D. Fu, and J. Liu, "Towards bridging online game playing and live broadcasting: design and optimization," in *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2015, pp. 61–66.

[126] S. Ahmad, C. Bouras, E. Buyukkaya, R. Hamzaoui, A. Papazois, A. Shani, G. Simon, and F. Zhou, "Peer-to-peer live streaming for massively multiplayer online games," in *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 67–68.

[127] F. Chen, C. Zhang, F. Wang, J. Liu, X. Wang, and Y. Liu, "Cloud-assisted live streaming for crowdsourced multimedia content," *Multimedia, IEEE Transactions on*, vol. 17, no. 9, pp. 1471–1483, 2015.

[128] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal *et al.*, "Quantitative comparisons of the state-of-the-art data center architectures," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1771–1783, 2013.

[129] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[130] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD*

international conference on Management of data. ACM, 2008, pp. 1099–1110.

[131] "Apache HBase," http://hbase.apache.org/.

[132] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, vol. 8, 2010, p. 9.

[133] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[134] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, 2011.

[135] "Scalable machine learning and data mining," http://mahout.apache.org/.

[136] "Apache Sqoop," http://sqoop.apache.org/.

[137] "Apache Flume," https://flume.apache.org/.

[138] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[139] "Apache Storm," http://storm.apache.org/.

[140] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache tez: A unifying framework for modeling and building data processing applications," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1357–1369.

[141] "The Hadoop Ecosystem Table," http://hadoopecosystemtable.github.io/.

[142] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI, "Making sense of performance in data analytics frameworks," in *Proc. ACM NSDI*, 2015.

[143] F. McSherry, "The impact of fast networks on graph analytics ," http://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2015-07-08-timely-pagerank-part1.html, 2015.

[144] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NetAgg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proc. ACM CoNEXT*, 2014.

[145] O. OMalley, "Terabyte sort on apache hadoop," *Available online at: http://sortbenchmark.org/Yahoo-Hadoop.pdf*, 2008.

[146] R. Nambiar, *Benchmarking Big Data Systems: Introducing TPC Express Benchmark HS*, 2015.

[147] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proc. ACM SIGMOD*, 2009.

[148] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the mapreduce-based data analysis," in *Proc. IEEE Data Engineering Workshop*, 2010.

[149] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proc. IEEE ICDM*, 2009.

[150] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *Proc. IEEE FSKD*, 2010.

[151] S. B. Yoginath and K. S. Perumalla, "Efficient parallel discrete event simulation on cloud/virtual machine platforms," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 26, no. 1, p. 5, 2015.

[152] H. Yang, Z. Luan, W. Li, and D. Qian, "Mapreduce workload modeling with statistical approach," *Journal of Grid Computing*, vol. 10, no. 2, pp. 279–310, 2012.

[153] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in MapReduce setups," in *Proc. IEEE MASCOTS*, 2009.

[154] J. Tan, X. Meng, and L. Zhang, "Performance analysis of coupling scheduler for mapreduce/hadoop," in *Proc. IEEE INFOCOM*, 2012.

[155] S. Frølund and P. Garg, "Design-time simulation of a large-scale, distributed object system," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 4, pp. 374–400, 1998.

[156] T. Li and J. Liu, "Cluster-based spatiotemporal background traffic generation for network simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 25, no. 1, p. 4, 2015.

[157] Y. Qiao, X. Wang, G. Fang, and B. Lee, "Doopnet: An emulator for network performance analysis of hadoop clusters using docker and mininet," in *Computers and Communication (ISCC), 2016 IEEE Symposium on.* IEEE, 2016, pp. 784–790.

[158] Z. Xie, Z. Cao, Z. Wang, D. Zang, E. Shao, and N. Sun, "Modeling traffic of big data platform for large scale datacenter networks," in *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on.* IEEE, 2016, pp. 224–231.

[159] K. Bilal, S. U. Khan, and A. Y. Zomaya, "Green data center networks: challenges and opportunities," in *Frontiers of Information Technology (FIT), 2013 11th International Conference on.* IEEE, 2013, pp. 229–234.

[160] D. E. Comer and R. E. Droms, *Computer networks and internets.* Prentice-Hall, Inc., 2003.

[161] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data networks.* Prentice-Hall International New Jersey, 1992, vol. 2.

[162] S. Garg and M. Kappes, "An experimental study of throughput for udp and voip traffic in ieee 802.11 b networks," in *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, vol. 3. IEEE, 2003, pp. 1748–1753.

[163] "Cisco Data Center Infrastructure 2.5 Design Guide," http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf.

[164] "Juniper Data Center LAN Connectivity Design Guide," http://docs.media.bitpipe.com/io_10x/io_105187/item_544348/Data%20Center%20LAN%20Connectivity%20Design%20Guide.pdf.

[165] "Rethinking the data center network," http://www.infoworld.com/article/2608992/data-center/data-center-rethinking-the-data-center-network.html.

[166] "TR10-Software-Defined-Networking," http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/.

[167] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[168] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[169] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[170] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *Communications Magazine, IEEE*, vol. 51, no. 11, pp. 24–31, 2013.

[171] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[172] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 183–197.

[173] "Introducing data center fabric, the next-generation Facebook data center network," https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[174] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[175] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "Mdcube: a high performance network structure for modular data center interconnection," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 25–36.

[176] P. Costa, A. Donnelly, G. Oshea, and A. Rowstron, "Camcube: a key-based data center," *Microsoft Res., Redmond, WA, USA, Technical Report MSR TR-2010-74*, 2010.

[177] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 75–86, 2008.

[178] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *INFOCOM 2009, IEEE.* IEEE, 2009, pp. 2276–2285.

[179] L. Gyarmati and T. A. Trinh, "Scafida: A scale-free network inspired data center architecture," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 5, pp. 4–12, 2010.

[180] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 225–238.

[181] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Co-NEXT '10 Proceedings of the 6th International COnference.* ACM, 2010, p. 16.

[182] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *Communications (ICC), 2014 IEEE International Conference on.* IEEE, 2014, pp. 3106–3111.

[183] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[184] R. S. Couto, M. E. M. Campista, and L. H. M. Costa, "A reliability analysis of datacenter topologies," in *Global Communications Conference (GLOBECOM), 2012 IEEE.* IEEE, 2012, pp. 1890–1895.

[185] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 4, pp. 397–413, 1993.

[186] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[187] S. Floyd, "Tcp and explicit congestion notification," *ACM SIGCOMM Computer*

Communication Review, vol. 24, no. 5, pp. 8–23, 1994.

[188] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "Rem: Active queue management," *IEEE network*, vol. 15, no. 3, pp. 48–53, 2001.

[189] N. K. Jaiswal, *Priority queues*. JSTOR, 1968, vol. 50.

[190] J. Nagle, "On packet switches with infinite storage," 1985.

[191] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4. ACM, 1989, pp. 1–12.

[192] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management," in *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 123–134.

[193] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A. Foong, "Tcp onloading for data center servers," *Computer*, no. 11, pp. 48–58, 2004.

[194] K. Park and T. Tuan, "Performance evaluation of multiple time scale tcp under self-similar traffic conditions," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 10, no. 2, pp. 152–177, 2000.

[195] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath tcp," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 10.

[196] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 63–74, 2011.

[197] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115–126, 2012.

[198] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[199] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar,

and M. Seaman, "Data center transport mechanisms: Congestion control theory and ieee standardization," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on.* IEEE, 2008, pp. 1270–1277.

[200] P. Devkota and A. Reddy, "Performance of quantized congestion notification in tcp incast scenarios of data centers," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on.* IEEE, 2010, pp. 235–243.

[201] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on.* IEEE, 2010, pp. 58–65.

[202] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.* ACM, 2015, pp. 523–536.

[203] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats *et al.*, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.* ACM, 2015, pp. 537–550.

[204] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[205] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data-center networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 2, pp. 345–358, 2013.

[206] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[207] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Transactions on*

Parallel and Distributed Systems, vol. 27, no. 11, pp. 3366–3380, 2016.

[208] V. Paxson, "Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 5, pp. 5–18, 1997.

[209] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 111–122.

[210] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1. ACM, 2004, pp. 392–392.

[211] "Playing video games is a full-time job for this twitch streamer," http://mashable.com/2014/03/27/twitch-streamers/.

[212] "Thegamesdb.net," thegamesdb.net.

[213] "Giantbomb.com," www.giantbomb.com/api.

[214] "Youtube videos viewed in 1 second," http://www.internetlivestats.com/one-second/#youtube-band.

[215] M. Wattenhofer, Y. Interian, J. Vaver, and T. Broxton, "Catching a viral video," in *Proceedings Workshop on Social Interaction Analysis and Service Providers*, 2010.

[216] "How sony is enlisting gamers on twitch, reddit to develop new titles," http://phys.org/news/2014-08-sony-gamers-twitch-reddit-titles.html.

[217] "Metacritic.com," http://www.metacritic.com/game.

[218] "Metacritic alexa rank," http://www.alexa.com/siteinfo/metacritic.com.

[219] "Is metacritic ruining the games industry," http://uk.ign.com/articles/2012/07/16/is-metacritic-ruining-the-games-industry.

[220] "Twitch now live streams game developers as they code," http://www.tubefilter.com/2014/10/31/game-development-category.

[221] G. Nencioni, N. Sastry, J. Chandaria, and J. Crowcroft, "Understanding and decreasing the network footprint of over-the-top on-demand delivery of tv content," in *Proc. WWW Conference*, 2013.

[222] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the Video

Popularity Characteristics of Large-scale User Generated Content Systems," vol. 17, no. 5, pp. 1357–1370, 2009.

[223] N. Sastry, "How to tell head from tail in user-generated content corpora," in *Proc. Conference on Weblogs and Social Media*, Dublin, Ireland, June 2012.

[224] H. Abrahamsson and M. Nordmark, "Program popularity and viewer behaviour in a large tv-on-demand system." New York, NY, USA: ACM, 2012, pp. 199–210. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398798

[225] "Top 100 largest overall prize pools of tournaments," http://www.esportsearnings.com/tournaments.

[226] "League of legends champions crowned: Samsung white wins e-sports tournament," http://www.ibtimes.com/.

[227] A. Hern, "Twitch plays pokémon: live gamings latest big hit," *The Guardian*, 2014.

[228] H. Yin, X. Liu, F. Qiu, N. Xia, C. Lin, H. Zhang, V. Sekar, and G. Min, "Inside the bird's nest: measurements of large-scale live vod from the 2008 olympics," in *Proc. of ACM IMC*, 2009, pp. 442–455.

[229] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "On measuring the client-side dns infrastructure," in *Proceedings of the 2013 conference on Internet Measurement (IMC '13)*. ACM, 2013, pp. 77–90.

[230] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure," in *Proceedings of the 2013 ACM Conference on Internet Measurement (IMC '13)*. ACM, 2013, pp. 313–326.

[231] "PeeringDB - AS46489 Twitch.tv," https://www.peeringdb.com/net/1956.

[232] "AS46489 Twitch.tv IPv4 Peers," http://bgp.he.net/AS46489#_peers.

[233] R. Fanou, G. Tyson, P. Francois, A. Sathiaseelan *et al.*, "Pushing the frontier: Exploring the african web ecosystem," in *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. International World Wide Web Conferences Steering Committee, 2016.

[234] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first

step towards comprehensive traffic forecasting in cloud computing," in *Proc. IEEE INFOCOM*, 2014.

[235] "The Apache Mahout project," http://mahout.apache.org/, 2017.

[236] "Statistical Workload Injector for MapReduce (SWIM)," https://github.com/SWIMProjectUCB/SWIM/wiki.

[237] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite," 2012.

[238] D. Cheng, P. Lama, C. Jiang, and X. Zhou, "Towards energy efficiency in heterogeneous hadoop clusters by adaptive task assignment," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on.* IEEE, 2015, pp. 359–368.

[239] H. Ahmed, M. A. Ismail, and M. F. Hyder, "Performance optimization of hadoop cluster using linux services," in *Multi-Topic Conference (INMIC), 2014 IEEE 17th International.* IEEE, 2014, pp. 167–172.

[240] K. Kambatla and Y. Chen, "The truth about mapreduce performance on ssds." in *LISA*, 2014, pp. 109–118.

[241] N. B. Rizvandi, J. Taheri, R. Moraveji, and A. Y. Zomaya, "Network load analysis and provisioning of mapreduce applications," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on.* IEEE, 2012, pp. 161–166.

[242] "Hadoop Cluster Setup," https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html.

[243] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *Proc. IEEE MASCOTS*, 2011.

[244] R. M. Fujimoto, "Research challenges in parallel and distributed simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 26, no. 4, p. 22, 2016.

[245] D. Lugones, K. Katrinis, M. Collier, and G. Theodoropoulos, "Parallel simulation models for the evaluation of future large-scale datacenter networks," in *Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th Interna-*

tional Symposium on. IEEE, 2012, pp. 85–92.

[246] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[247] S. Rao, D. Goswami *et al.*, "Ns3 simulator for a study of data center networks," in *Parallel and Distributed Computing (ISPDC), 2013 IEEE 12th International Symposium on*. IEEE, 2013, pp. 224–231.

[248] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 39–50.

[249] S. A. Jyothi, A. Singla, P. Godfrey, and A. Kolla, "Measuring and understanding throughput of network topologies," *arXiv preprint arXiv:1402.2531*, 2014.

[250] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, "Joint host-network optimization for energy-efficient data center networking," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 623–634.

[251] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The newreno modification to tcp's fast recovery algorithm," Tech. Rep., 2012.

[252] S. Zafar, A. Bashir, and S. A. Chaudhry, "On implementation of dctcp on three-tier and fat-tree data center network topologies," *SpringerPlus*, vol. 5, no. 1, p. 766, 2016.