VIDEO POPULARITY METRICS AND BUBBLE CACHE EVICTION ALGORITHM ANALYSIS

By

Hildebrand Weisenborn

A Thesis submitted for the degree of Doctor of Philosophy
School of Computer Science and Electronic Engineering
University of Essex
October 2017

Acknowledgements

Thanks to EPSRC and BT for supporting this work through an EPSRC CASE award title "Information-centric networking for video delivery over existing and emerging ISP networks."

A special thanks to my supervisor, Dr Martin Reed, and my family and friends who have supported me throughout my research.

Abstract

Video data is the largest type of traffic in the Internet, currently responsible for over 72% of the total traffic, with over 883 PB of data per month in 2016 [1]. Large scale CDN solutions [2–5] are available that offer a variety of distributed hosting platforms for the purpose of transmitting video over IP. However, the Internet Protocol (IP) protocol, unlike Information Centric Networking (ICN) protocol implementations, does not provide an any-cast architecture from which a CDN would greatly benefit. In this thesis we introduce a novel cache eviction strategy called "Bubble," as well as two variants of Bubble, that can be applied to any-cast protocols to aid in optimising video delivery. Bubble, Bubble-LRU and Bubble-Insert were found to greatly reduce the quantity of video associated traffic observed in cache enabled networks. Additionally, analysis on two British Telecom (BT) provided video popularity distributions leveraging Kullback-Leibler and Pearson Chi-Squared testing methods was performed. This was done to assess which model, Zipf or Zipf-Mandelbrot, is best suited to replicate video popularity distributions and the results of these tests conclude that Zipf-Mandelbrot is the most appropriate model to replicate video popularity distributions. The work concludes that the novel cache eviction algorithms introduced in this thesis provide an efficient caching mechanism for future content delivery networks and that the modelled Zipf-Mandelbrot distribution is a better method for simulating the performance of caching algorithms.

Contents

List of Figures

List of Tables

Acronyms

IP Internet Protocol

P2P Peer-to-Peer

CDN Content Delivery Network

WWW World Wide Web

VoD Video On Demand

ICN Information Centric Networking

CCN Content Centric Networking

NDN Networking Named Content

DONA Data-Orientated Network Architecture

PURSUIT Publish-Subscribe Internet Technology

PSIRP Publish-Subscribe Internet Routing Paradigm

LCE Leave Copy Everywhere

SAIL Scalable & Adaptive Internet soLutions

COMET COntent Mediator architecture for content-aware nETworks

RAND Random

LRU Least Recently Used

FIFO First-in First-out

LFU Least Frequently Used

LFUDA Least Frequently Used with Dynamic Aging

Perfect-LFU Perfect Least Frequently Used

VoDu VoD-time-unlimited

UGC User Generated Content

BT British Telecom

SLA Service Level Agreement

KL Kullback-Leibler

PCS Pearson Chi-Squared

PCC Pearson Correlation Coefficient

PDF Probability Density Function

BU Boston University

HTML Hypertext Markup Language

DDoS Distributed Denial of Service

CDF Cumulative Distribution Function

OSI Open Systems Interconnection

PSIRP Publish-Subscribe Internet Routing Paradigm

BF Bloom Filters

URL Uniform Resource Locator

PhD Philosophiae Doctor

DNS Domain Name System

TCP Transmission Control Protocol

UDP User Datagram Protocol

1

Introduction to Thesis

Video On Demand is vastly expansive and makes up a large quantity of the total traffic on the internet. Global IP video traffic is predicted to constitute 82% of all consumer internet traffic by 2020, up from 70% in 2015 [6] and 72% in 2016 which equates to approximately 883 PB per month [1]. As Video on Demand becomes increasingly more prevalent the research community has sought ways to alleviate the strain video request and data delivery has on a network. To contribute to this effort, a great understanding of Video request data is required, as well as strategies as to how one may achieve alleviation of strain of video data on a network.

1.1 Thesis Motivation and Contribution

As previously stated in the introduction of this Chapter, Video has become a dominant source of traffic on the Internet. This Thesis aims to provide analysis of video request data, as well as introduce methods one can use to achieve a reduction in video traffic on a network without penalties to performance or quality, using technologies that leverage caching in nodal computer networks.

Video Request Data is complex and intricate and is responsible for a tremendous amount of strain on most consumer-facing network infrastructures. If a naive perspective were to be presented, the total amount of requests made to a Video Distribution Platform can be equated and used to assess how many video objects are delivered to users from a Video Delivery System. This approach appears to be the most intuitive method of observing video traffic. However, it does not include important details such as segmentation, the distributed Content Delivery Network (CDN) servers, and the routing that occurs to ensure that the video data reaches the distributed users. Observations of video popularity distributions [7–18] are typically made over time and then concluded to be such. This means that the items measured are frequently introduced before the observations start or remain to receive requests after the observations end, meaning that data is often incomplete. Although not all these errors in observation are addressed and tackled with structurally better observations in this Thesis due to a limited dataset, a new video request generator is proposed in an effort to encompass and overcome observed shortcomings in other simulation environments and observed data-sets that are available today, which bear the constraints that those who came before were subjected to during development. The request generator provides a request generation platform that accounts for changing popularity in items over time with the introduction of decay and a changing data-set over time that retires items and introduces new ones.

Video Request Data is often measured through the popularity index of each of the

items observed in the system. This popularity distribution has caused discussion in the research community [14] largely due to the lack of data currently available to researchers. Additionally, the methods of analysing video request distributions appear overly simplified, often using a see-by-eye test to describe two sets of data, one generated and one observed, side-by-side to be similar enough to consider them approximately equal. In this Thesis, two individual VoD data-sets provided by British Telecom (BT), the sponsor of this Philosophiae Doctor (PhD) and Thesis, aid to help us discover which popularity distribution suggested in the research community shares the greatest resemblance to real video request data. A number of different analytical tests and simulations were performed to rule out bias as to what generated popularity distribution best matches the video request data of a real VoD platform. The tests used were a combination of Pearson Chi-Squared, Pearson Correlation Coefficient and Kullback-Leibler which individually provided a method to confirm what model was more appropriate in replicating an existing data-set. The models considered as appropriate replications models for the BT data-sets were the Zipf-Mandelbrot model and the Zipf-Like model. To conclude, Zipf-Mandelbrot was demonstrated to be more appropriate for modelling a video request distribution such as the observed video request distributions provided by BT.

The technologies used to currently distribute video content are large scale CDNs. Websites such as YouTube [19], owned by Google, house their own CDN infrastructure however most VoD providers use CDN services offered by companies such as Akamai [3], Cloud Flare [4] and Microsoft Azure [5]. These providers deliver content to consumers in a quick and secure manner by delivering data from a server local to the user. A CDN provider, such as Microsoft Azure, outsources traffic managing to a Traffic Manager to assure a server that is local to the user, responds to the user request. Other services may distribute their servers and hand off the forwarding of requests to a local server to Domain Name System (DNS), forwarding [2] solutions implemented by companies such as Microsoft. Akamai handles load balancing by using a combination of application layer awareness and DNS forwarding. All these methods are used to ensure users are

served data locally to their own physical location, a challenge not handled by the classic IP infrastructure, but by the application layer and network layer in the case of the Akamai infrastructure and their Modern Load Balancing examples. This innovation in load balancing and traffic management demonstrated in Azure and Akamai, additionally demonstrates the constraints inherent to the current infrastructure of the Internet. Many of the mentioned constraints of distributed servers providing the same content in different geo-locations can be solved with modern alternatives to IP and Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) with an any-cast architecture such as ICN. These architectures are introduced and discussed in the Thesis and, additionally enable the possibility of seamless distributed networks and enable caching functionality.

Caching is often suggested as a method of reducing traffic associated with frequently requested objects on a network, as well as frequently performed operations in a processor. It also helps alleviate the amount of operations required and optimises the performance of many hardware and software infrastructures. Caching can also be leveraged specifically to alleviate a network of a large quantity of traffic when applied to video delivery, as video request data exhibits repetitive behaviours as the frequency of requests for items is often heavily skewed, majoritively only towards a small subset of items [9, 10, 12], favouring local storage as being able to cache frequently requested objects. This Thesis introduces cache eviction algorithms that can be applied to a cache enabled network. The eviction algorithms are a method of caching that operate with no knowledge of the network and its state, but rather operate in an inexpensive manner that, in some scenarios, rivals the more well known cache eviction algorithms; namely LRU and LFU.

The algorithm introduced in this Thesis is the Bubble cache eviction algorithm. A patent application was submitted by BT and later rejected on the premise that prior art [20] was found to contain a switching mechanism similar to the mechanism used in

the Bubble algorithm. The creating and submitting of the patent application to the European Patent Office took upwards of a year, thus delaying the option to submit a publication introducing the Bubble algorithm to the wider research community.

1.2 Thesis Outline

This Thesis is structured in the manner one may assume to follow when aiming to improve video delivery over a cache enabled network.

Chapter 2 introduces technologies and methods associated with caching and video delivery systems. A number of different aspects of video delivery will be explored such as; the year of observation, user counts, requests per day, requests per user, locations of observation and the number of unique videos; all within diverse VoD systems. Identifying these characteristics as well as other areas of research that impact local caching and the technologies associated with such systems will also be explored here.

Chapter 3 addresses the shortage in evaluation of video-focused consumption models. While there has been conjecture that Zipf-Mandelbrot is a good model, this work shows, through empirical data, that consumption patterns do indeed comply to this distribution. This is achieved by analysing two example, large-scale, empirical datasets, provided by BT, as well as using synthetically generated consumption data, following Zipf and Zipf-Mandelbrot distributions. These models include standard testing methods such as; Pearson chi-square, Pearson's correlation-coefficient, as well KL divergence. This Thesis' study demonstrates that Zipf-Mandelbrot better fits realistic consumption data than the previously most widely considered model, the Zipf-like distribution. Furthermore, the study shows that the expected behaviour of cache-enabled video delivery systems is closer to that of the empirical dataset when using the Zipf-Mandelbrot model than the Zipf-like model.

Chapter 4 assesses video delivery systems closely to outline the required data to more closely simulate a video delivery system. This chapter introduces a method by

which one can create pseudo-real requests that demonstrate time based characteristics in an effort to achieve a greater understanding of the effects of video decay, introduction and removal. This additionally aids in concluding that the current method of observing video request data is flawed as the passage of time over the period of observation can have many effects on the popularity distributions experienced.

In Chapter 5 a novel eviction algorithm is introduced. The algorithm is called the Bubble cache eviction algorithm, which is in reference to the Bubble sort algorithm, [21] which it resembles in some of its behaviours. The introduction of Bubble, together with the introductions of variations of Bubble, are intended to provide an alternative set of algorithms to the known cache eviction algorithms such as LRU, FIFO, Random (RAND) and LFU. Investigated in this chapter is the Bubble algorithm and its variations and the results one can expect in a variety of simulated VoD systems, as well as analysis investigating the effectiveness of the introduced algorithms.

2

Background Research

VoD systems vary and are diverse in characteristics and behaviours [9, 10, 14, 17, 18, 22]. The years of observation, user counts, requests per day, requests per user, locations of observation and the number of unique videos are the gross level of varying characteristics when expressing the diversity of VoD systems currently live. Identifying these characteristics as well as other areas of research that impact local caching and the technologies associated with such systems will be explored in this Chapter. An additional focus of the study is that of technologies that would provide a platform from which video delivery could be seen to express a lesser strain with the option of localised caching, currently not natively possible in IP.

2.1 Video Delivery System Characteristics

2.1.1 Video Segmentation in the context of caching

Caching whole, uninterrupted multimedia objects can be very inefficient in network caching. This is due to the size of the multimedia objects and the sections of the multimedia objects that are consumed by the end-users. Video files could reach the size of a number of gigabytes and a large number of consumers may only watch the first $\frac{1}{4}$ of video. A method to avoid unnecessary latency and storage of multimedia objects is through segmentation. Segmentation suggests splitting the original multimedia objects into a number of smaller objects which are stored and treated as unique objects in the cache-enabled system/network. A number of varying strategies have been developed in an effort to minimise the latency experienced on a network and reduce the quantity of resources occupied in, for example, a cache of limited size. Key indicators were: Byte-hit ratio, deduction of requests with delayed start.

2.1.1.1 Fixed Segmentation

Fixed Segmentation splits the multimedia objects into a multitude of fractions evenly distributed to fill segments homogeneously. The fixed segmentation method has one clear benefit over Pyramid and Skyscraper segmentation regimes as it does not require segments to be valued separately. A cache eviction algorithm does not require knowledge of the size of the segment to increase the byte-hit ratio and performance of a cache, as would be required for Pyramid and Skyscraper segmentation schemes.

2.1.1.2 Pyramid Segmentation [23]

Pyramid segmentation postulates that the start of the multimedia object will most likely be requested and will remain most popular. The parts following the start are increasingly less important and thus kept in larger segments. The segments start at a fixed size at the beginning, exponentially increasing in size. This way the last half of the multimedia object can be removed with one single action. The first parts of the

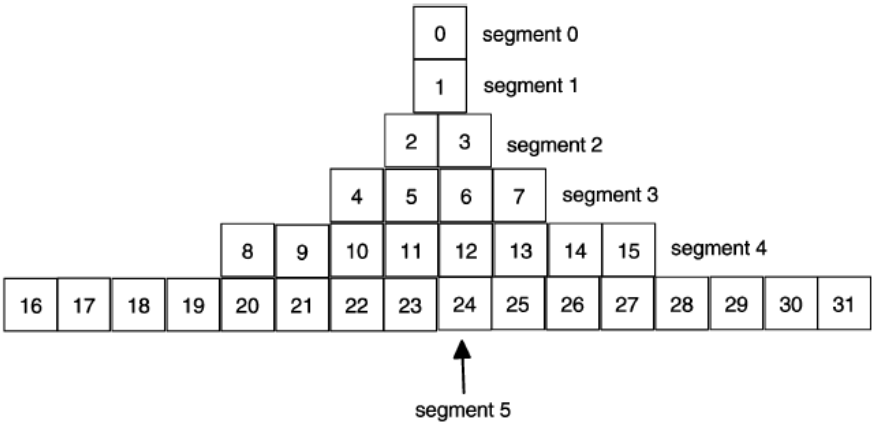


Figure 2.1: Segmentation of a video according to the Pyramid segmentation regimen

video will be segmented to be easily interchangeable in a cache. N_i^p denotes the size of segment i of a multimedia object under the pyramid segmentation regimen. Assume b is a constant and $b > 1$.

$$N_i^p = \begin{cases} 1, & \text{if } i = 0 \\ (b-1)b^{(i-1)}, & \text{if } i > 0 \end{cases} \tag{2.1}$$

b decides the segment size increase with each $i + 1$. Figure 2.1 shows an example pyramid segmentation structure where $b = 2$, as shown in Equation 2.1. Each box in Figure 2.1 indicates an equal sized fraction of the original multimedia object.

2.1.1.3 Skyscraper Segmentation [24]

A criticism of pyramid segmentation is that some segments grow to be very large. This may be detrimental to caches and user devices, such as set-up boxes, as storage is limited. A suggested solution to this problem is skyscraper segmentation which, as pyramid segmentation, increases the size of each subsequent segment, but more gradually. N_i^p denotes the segment size of segment i of a multimedia object under the Skyscraper regimen.

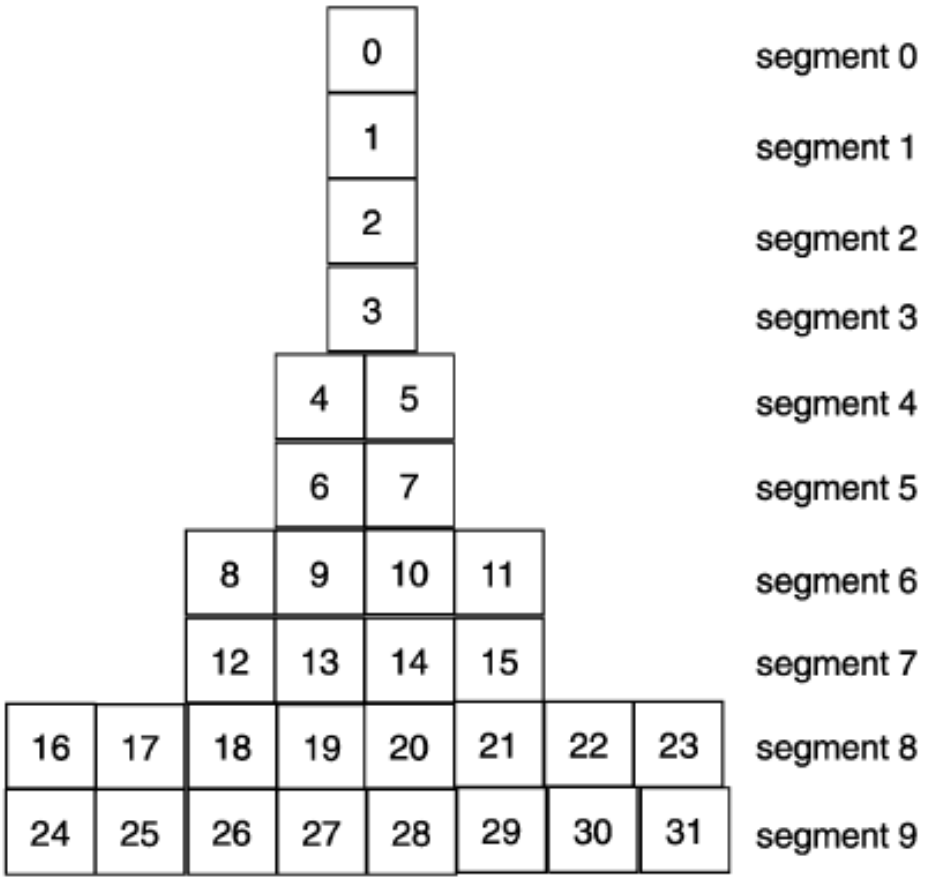


Figure 2.2: Segmentation of a video according to the Skyscraper segmentation regimen

$$N_i^p = \begin{cases} 1, & \text{if } i = 0, 1, , n \\ (b-1)b^{\left(\frac{i}{(n+1)}\right)-1} & \text{if } i > n \end{cases} \tag{2.2}$$

b and n decide the segment size increase with each $i+1$ in a sky scraper segmentation scheme, such as is described in Equation 2.2. Figure 2.2 shows an example skyscraper segmentation structure with $b = 2$. Each box in Figure 2.1 indicates an equal size fraction of the original multimedia object. Each row symbolises a single chunk on data that may be cached. n reduces the exponential increase that we experience in the pyramid segmentation scheme visible as repeating segment sizes for not just the first number of segments.

The reduction in segment size means local storage devices and network storage

devices can more easily handle the larger segments when storage is of limited capacity.

2.1.1.4 Cache Admission Policy for Segmentation [23]

A cache admission policy can be used, before object submission to a cache to function, as a pre-selection process. A Cache Admission Policy would be required for a segmentation scheme, such as Pyramid or Skyscraper segmentation schemes, as not each segment can be considered equally due to the range in size of each segment. Segment i should only replace segment j in a cache if i experiences a greater byte-hit ratio in the cache than j in the same time frame. This may be possible to determine in a Perfect Least Frequently Used (Perfect-LFU) enabled cache as the total requests for items not cached are recorded and considered when selecting items for caching, however most other cache eviction algorithms would be challenged to account for varying segment size.

2.1.2 Media and WWW Popularity Distributions

Many online media types have been identified to follow the same or similar trends in the frequency of requests. The objects have many different formats from Hypertext Markup Language (HTML) pages to images or videos. A number of studies have described a Zipf-like distribution to be a very closely fitting distribution to most generalised content observed through proxy servers available on the World Wide Web [25–28]. A simple see-by-eye test indeed confirms that a Zipf-like distribution can be observed in the recorded proxy data as show in Figures 2.3 [25], 2.4 and 2.5.

Almeida et al. [25] does suggest a Zipf distribution can be considered present, as is illustrated in Figure 2.3, as the data is shown on a log-log scale to be depicted as an almost straight line, which is a Zipf-like property. Almeida et al. [25] subjects the original data to a cache which measures cache-hit ratio by arbitrarily selecting objects for request according to the popularity distribution observed in the Boston University (BU) servers. The same test is performed for the Zipf distribution as items are arbitrarily selected with probabilities decided by the generated Zipf distribution.

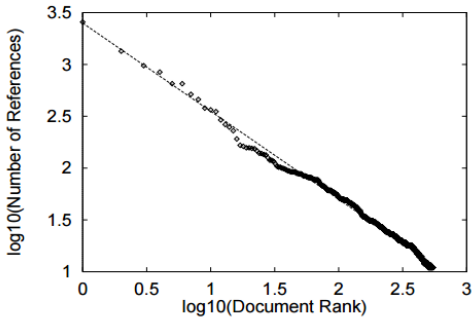


Figure 2.3: Zipf a = 1. Zipf Law Applied to Web Documents

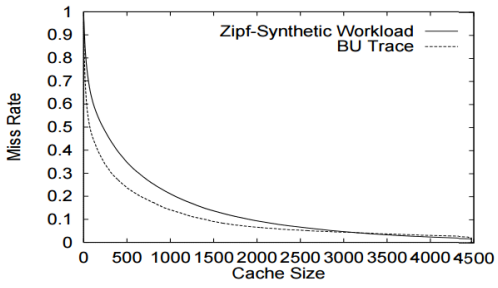


Figure 2.4: Miss rate for a Zipf-based synthetic workload and for an actual trace

Figure 2.4 illustrates the cache-hit ratios observed for a range of cache sizes. In a scenario where each distribution can be considered equal, the two lines for each Zipf and the BU trace would sit exactly atop each other, however, it is clear that the two observed cache-hit ratios are indeed different. The Zipf-based model is shown to be nearly 33% less likely to receive a cache hit rate with a cache size of 400 than the original BU dataset that is suggested to be an accurate hit-rate.

Breslau et al. [26] performed a fitting using the MatLabs curve fitting tool, excluding the top 100 items. The top 100 times are arguably the most influential items in the distribution as they are responsible for the 52% of requests of all the documents. The most probable reason for excluding the 100 most popular documents when fitting is due to the erratic nature of the most popular documents. Breslau et al. [26] do indicate that Zipf is indeed a model that closely resembles the original proxy popularity distribution which includes the 100 most frequently requested items as is illustrated in the graph Figure 2.5 [26].

Cunha et al. [27], illustrated in Figure 2.6 [27], Perform R-Squared testing to confirm that a Zipf Distribution is present in the observed object request distribution. The results suggest an exact fit to a Zipf-like distribution with a value of 1.00. The objects observed are categorised into a number of different groups such as, HTML, Images, Sound, Video, Text, Formatted Documents, Archive and other files. Image files are the great majority of requests, a result of many images contained in a single HTML file.

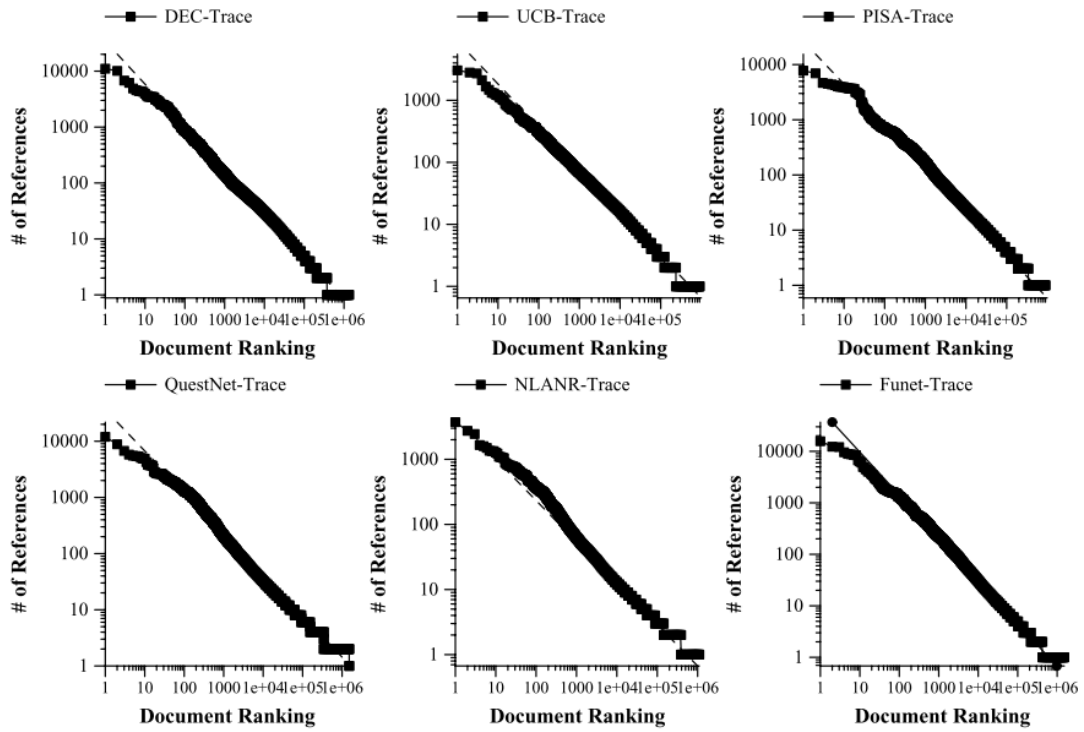


Figure 2.5: Zipf = 0.66 $\tilde{0}$.83 Web Caching and Zipf-like Distributions: Evidence and Implications

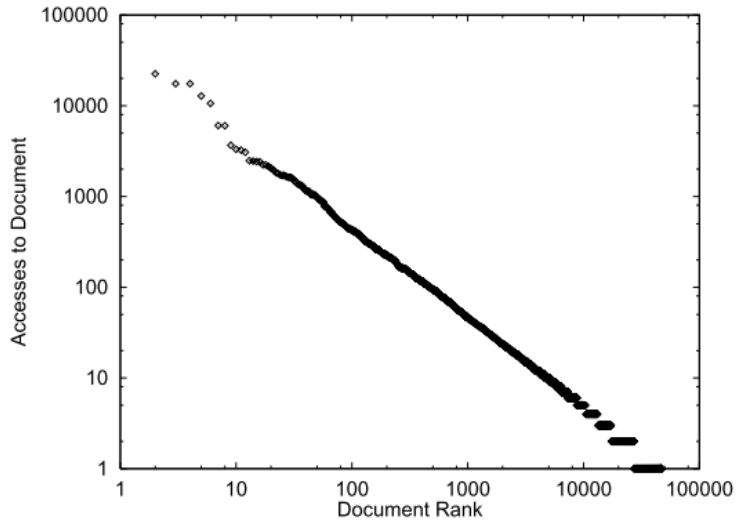


Figure 2.6: Zipf a=1, Characteristics of WWW client-based traces

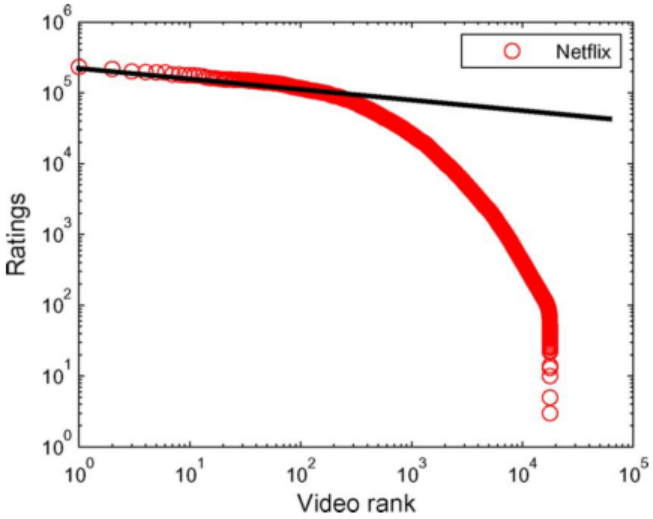
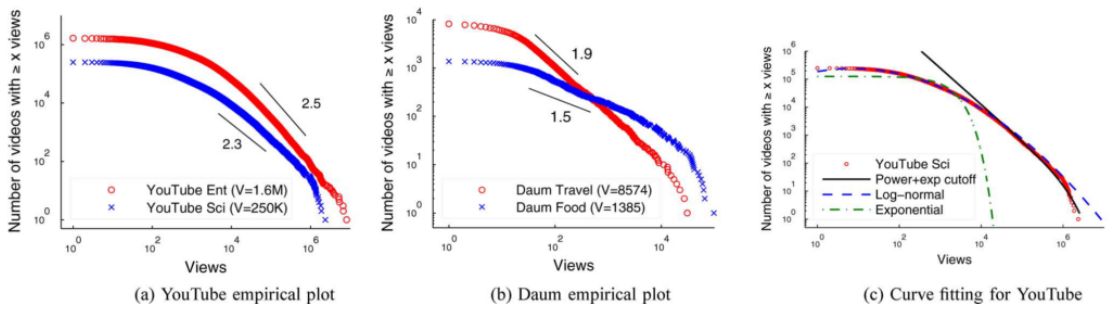


Figure 2.7: The Rank vs Rating of videos in the Netflix system pre-2006

2.1.2.1 Video Popularity Distributions

There seems to be majority consensus [7–18, 29, 30] that Zipf-like distributions are most representative of VoD systems. The main consensus originates from the observations first performed on general online systems and not exclusively video systems [25], [27], [31] which was later applied to VoD, assuming the two are interchangeable in regards to the observed popularity distributions. A number of studies [12, 16, 17] have expressed caution assuming that the popularity distribution of video content is Zipf-like, due to the heavy tail and the flat head seen on a log-log scale. Cha et al. [17] suggest that perhaps the truncation of the heavy tail is caused by the system implementation (Netflix [32]), which does not make niche content easily accessible. However, analysis of other studies [9, 10, 14, 18, 22] show that the truncated tail, and the flat head, appear in many other VoD systems thus opening up discussion on what model can be used to represent VoD popularity distributions accurately. Just as the Zipf-Mandelbrot and Zipf-like power-law distributions are frequently considered an appropriate model for video request data, a log-normal distribution may provide an equally appropriate model. The log-normal distribution has frequently demonstrated to be almost the same as power-law models [17, 33, 34].



(a) YouTube empirical plot (b) Daum empirical plot (c) Curve fitting for YouTube

Figure 2.8: UGC video request data. Views vs. Video Rank

Alternatively, Tang et al. [14] suggests that Zipf-Mandelbrot represents VoD popularity distributions more accurately, which have also been witnessed in comparable systems, such as P2P [15]. This trend may not be exclusive to video content but rather general multimedia-based content [12]; thereby, showing Zipf-Mandelbrot to be emerging as an alternative distribution model for describing general multimedia-based content (including video), as opposite to the conventional Zipf model. Though there is evidence that Zipf-Mandelbrot demonstrates to be an appropriate alternative to Zipf-like when modelling the popularity distributions seen on VoD systems, the evidence is limited as it largely has been researched to model multimedia systems which show resemblance to VoD instead of pure VoD systems.

2.1.2.2 User Generated Content (UGC) Videos

User Generated Content (UGC) is a large section of the total video traffic with the emergence of large UGC video delivery platforms such as YouTube, Daum, Vimeo, Vine and many more. UGC is content that is generated and consumed by a large amount of producers and consumers and typically contains videos which are more ephemeral, unpredictable and numerous [33]. UGC is very diverse and thus creates an environment where individuals will choose to watch a more diverse group of videos which is unlike previous Video Delivery Systems such as traditional television where focus of a large group of viewers was decided by a small group of TV channels, keeping focus towards a smaller sections and variation of video content.

Video popularity distributions experienced in UGC systems, such as the one shown in Figure 2.8 [33], demonstrates a longer tail and a smaller set of extremely popular videos relative to the total number of videos. In a Zipf-Like distribution this would appear as a distribution with $\alpha > 0.8$ such as an alpha variable as suggested in [25] and [26].

2.1.3 Video Popularity Decay

Video Popularity Decay has been discussed about in a range of research [16, 35–37], however it appears to rarely be quantified or analysed beyond the effects of the specific tests performed by the researchers in question. Additional to the lack of analysis of video decay, is the frequent omission of growth of popularity of video objects.

The subject of decay is discussed by Gummadi et al. [16] in the context of a peer-to-peer system called "Kazaa" in a rather poor manner as items larger than 100 MB are declared to experience a larger amount of request for items older than one month by 72% and items smaller than 10 MB are declared to experience a larger amount of requests towards items older than one month but by only 52% with the omission of medium sized items. The decay here is for multimedia items, not exclusively VoD, which causes some doubt as to how much this data resembles exclusively VoD data. It may well be possible that the majority of items observed are not video, dictating the decay-rate experienced by items in the system.

The subject of decay is also discussed by Li et al. [36] and Chen at al. [38] appear to agree that within their observed datasets that the initial day of observations see the videos decrease in popularity by 20%, and again decrease by 20% 9 days later. This observation is based in video request data that is explicitly not UGC on both Peer-to-Peer (P2P) based systems, as well as CDN run systems, such as Hulu and PPLive. Li et al. [36] appear to consider items once they are stored on the cloud device and then observed the decreasing popularity. This comes after a video is identified as popular enough to be submitted to the cloud, which may see a large part of the item's lifetime

be omitted (an hour minimum due to design), as the time before the item became popular it was not yet considered for measurement of popularity, especially considering the initial surge in popularity.

Chen et al. [39] follow the lifetime of video items in a system which includes the decay. The observation is made in a VoD service run by Tencent, which is one of the largest VoD services in China. It provides video to an active user base of over 50 million people. For the purpose of measuring video decay, videos are separated into categories, easily distinguishable by key characteristics (Movies, Music Videos, TV content, News and Sports), for which each category has a total decay rate measured over the duration of 7 days. The observations made suggest that most categories appear to see a great quantity of requests on the day of release, as well as the day following. After this period all categories see a drastic decrease in popularity, with the exception of the "Movies" category which appears to see a peak in interest on the third day the item is in the system. It is unclear if items on day 0 (the initial day of release) were in the system for the entirety of it, or just the later section of the day. This additional bit of missing information would point out if the items, almost immediately after release, start seeing a decreasing request rate or if this happens, as suggested by Chen et al., on the second day in the system. The conclusive statement by Chen et al. [39] is that video items in their infancy receive a great amount of promotion, mainly on the front-page of the VoD application, which may be the primary reason for the popularity received by these newly introduced video items.

Avramova et al. [37] provide a close look at the decay of video popularity of primarily video items which have already achieved a great deal of popularity by their submission to the top 50, list on their respective platforms, such as YouTube (US and JP platforms), IMDB rental records, and "Uitzending Gemist(.nl)" which is a Dutch TV catch-up service. Once submitted to the top 50 the items were observed for a minimum of 30 days. The observations concluded that it is primarily the UGC that sees a thick tail in the observed decay curve which can be described as a power-law distribution. All other

VoD System	Nr. of Observed Entries	τ	α	β
YouTube (Japan & USA)	≤ 160	≤ 20	≈ 0	0.5
UitZending Gemist	≤ 42	≥ 10	$\approx 10^6$	0.5
IMDB	≤ 18	≥ 10	$\approx 10^6$	0.5

Table 2.1: Results gathered by Avramova et al. The results are approximated from the graphs in the publication which can be factored in to the simulator

traces, described as TV catch-up services, appear to follow a more drastic decrease over time described as following exponential decay. The formula used by Avramova et al. [37] to plot the decay in the form of a Cumulative Distribution Function (CDF) experienced by video items in all systems is as shown in Equation 2.3

$$I_k(t) = \rho_k \left[1 - \left(1 + \frac{\left(\beta^{\frac{-1}{\alpha_k}} - 1 \right) (t - \Theta_k)}{\tau_k} \right)^{-\alpha_k} \right] \tag{2.3}$$

In Equation 2.3 [37] τ describes the time it takes for a fraction $1 - \beta$ of the total view count to accumulate. Δt describes the time at which the item is observed and Θ describes the time of entry in the system. α is the important variable to consider as it is the variables that is form-determining in the function. If α is large, the function produces an exponential curve which would see the item gain the majority of its requests in the beginning of its existence. If α is small, the function produces a power-law curve which would see the item remain relevant for an extended period of time, with a small amount of popularity remaining throughout the item's existence.

2.2 Any-cast protocols/systems

IP is the current protocol used exclusively on the internet at the Open Systems Interconnection (OSI) [40] network layer protocol. It provides a location based method of accessing machines and was designed between the 1960s and 70s. The problem IP aimed to resolve was the sharing of resources over a distance between machines. The method used to retrieve resources was to reach a machine in a specific location to then request

the resource, creating a location based request methodology. The internet in current days sees less interest in the physical location from which the content originates, but rather the content itself irrespective of the origin. In the current architecture of the internet we see a need for any-cast algorithms which do not discriminate based on the location of data, but instead focuses on the proximity of the data required. Any-cast functions on the premise that requests can originate from many or one source and be satisfied by many or one single source on the network. An architecture such as the ICN model has been conceived to provide a method of delivering requested data objects without emphasising on the location of the data. ICN functions at layer 3.5 of the OSI depending on the implementation.

2.2.1 ICN Implementations

There are a number of projects and exemplar implementations of ICN such as; Data-Orientated Network Architecture (DONA) [41], Publish-Subscribe Internet Technology (PURSUIT) [42, 43], Scalable & Adaptive Internet soLutions (SAIL) [44], COntent Mediator architecture for content-aware nETworks (COMET) [44] and Content Centric Networking (CCN), a US funded project that later became Networking Named Content (NDN). All of these architectures follow an ICN architecture of which the PURSUIT implementation is used in this Thesis to aid in assessing effectiveness of caches as the ICN architecture is uniquely able to utilise caches in an a manner considerate of the location of the content. Reed et al. [42] describes the PURSUIT the publish-subscribe Internet routing paradigm that is based on the Publish-Subscribe Internet Routing Paradigm (PSIRP) architecture which Fotiou et al. [45] and Trossen et al. [46] describe in further detail. In ICN the location is decoupled from the object, making it possible to request the object and route it in a matter different from the currently internet. One method used in the CCN architecture uses a Uniform Resource Locator (URL) referring to an object. The URL is closely bound to the location of the system on which the object is stored and made available from. Many implementations of ICN still use IP as an underlying model, however implementations have been developed

with an alternative forwarding model leveraging bloom Filters (Bloom Filters (BF)) [42, 47] which makes a source routing approach possible. The PSIRP model allows the PURSUIT implementation to leverage in-network caching, multi-party communication through replication, and an interaction model that decouples sender and receiver.

An example implementation of ICN may see an architecture where subscriptions of users are handled by a Rendezvous (replacing the previously mentioned requirement of a URL seen in CCN) which functions as the system linking Subscriber and Publisher when a match is possible - this implementation is seen in the PURSUIT project. Once matched, the Rendezvous forwards the information of the object and location of both Subscriber and Publisher to the Topology Manager which generates a BF which contains the instructions the Publisher can use to forward the object to the Subscriber. The BF is sent from the Topology Manager to the Publisher which can forward the object accordingly. In this example it can be seen that, as objects are forwarded along the network, a caching node may choose to store the requested object and make it available; ready for any future requests of that object to spare network resources in the future. The process through which an object is forwarded from an original publisher would not differ if the object was forwarded by a caching node in its stead from the perspective of a Subscriber or the Rendezvous.

2.3 Cache Conscious Routing Strategies

For the purpose of testing the behaviour of the different popularity distributions four routing strategies were selected based on the performance gained and their individual diversity and functional differences. Routing strategies dictate the flow of traffic in a network with focus on cache storage and reducing total traffic on the network. The strategies available are numerous and a few of them are introduced and discussed below:

2.3.1 Cache Less for More (CL4M)

Cache less for more [48] considers centrality to decide where along the request path items should be cached. Caching at nodes only with the highest betweenness centrality

means the cached items are not replicated unnecessarily to all nodes on the request path and have an increased chance of residing on a node which requests are likely to pass over.

2.3.2 Leave Copy Down (LCD)

Leave Copy Down [49] routing strategy works on the basis that items are replicated towards the edge of the network from the original source of the requested items. Items are replicated towards the requester one level down with each request. This process gradually pushes frequently requested items towards the outer edges of the network.

2.3.3 Probabilistic Caching (ProbCache)

Probabilistic caching [50] places items on the path between source and requester based on a number of characteristics of the path. ProbCache is an algorithm that considers caching capabilities and content capacity, path lengths, and multiplexes content flows accordingly.

2.3.4 Leave Copy Everywhere (LCE)

When an object is requested in a Leave Copy Everywhere (LCE) implemented network it is stored on each cache enabled node over which it is sent.

2.3.5 Hash-routing

The hash routing strategy [51] requires each cache enabled node on the network to implement a hash function. The hash of the content transmitted is mapped to a specific node with the same hash. The node to which the content is mapped to is the only node on the network able to cache that content. This means in some implementation of Hash-Routing objects are not forwarded along the shortest path to the subscriber, but instead forwarded to the cache enabled node. In some instances it is forwarded to the cache enabled node as well as to the subscriber. This method of caching avoids replication of items in the network.

2.3.5.1 Hash Hybrid Symmetrical-Multicast Routing

The Hash routing strategies [51] are based on the idea of hashing items and mapping them to caches in the network. This method of caching may mean caching items off their original path of requests but avoids item replication on the network. A copy will always be sent to the designated cache if the item does not yet exist on the relevant cache enabled node. The decision whether the item should be sent from the cache or the original source to the requester depends on the total path length between the two sources of the item and the original requester.

2.4 Cache Eviction Algorithms

Cache eviction policies dictate the content of the storage devices on which they are implemented. The storage device would, in most scenarios, be located on an intermediate link somewhere on a network. Cache enabled network devices select content that is requested and sent along the path on which they are located for storing. End devices may request data items located on the cache enabled network devices instead of content servers as content servers may be considerably further removed on the network than cache enabled devices with the same desired content. Depending on what cache eviction algorithm is implemented, a number of different actions can be taken by the cache in order to attempt to maintain the items in the cache that will receive the largest amount of requests in the future, thus reducing the traffic downstream from the caching device to the best potential. Four opportunistic cache eviction policies were considered for measuring the performance differences between Zipf and Zipf-Mandelbrot.

2.4.1 LRU

The items in Least Recently Used (LRU) that experience eviction are those which were requested least recently. Every requested item is added to the cache or, if already present in the cache, noted to be the most recently requested item. The operational

cost of LRU is $O(1)$.

2.4.2 LFU

The items in Least Frequently Used (LFU) that experience eviction are the ones with the smallest request count. Items are added by removing the items in the cache with the smallest number of observed requests. Item request counts are reduced every n number of requests to make old items available for eviction. LFU has a slightly higher operational cost of $O(c)$ where c represents the cache size when replacing items in the cache. A simple search is $O(1)$.

2.4.3 RANDOM

The random cache eviction policy evicts random items to make space for new items. This policy does not consider the frequency of requests of items held in the cache and does no prioritization of any items. Random has an operational cost of $O(1)$.

2.4.4 FIFO

First in First out (FIFO) works by arranging items in an ordered list and evicting items from the bottom and adding them on the top. FIFO and LRU share a very similar method of selecting items for eviction. FIFO does not move items to the top of the cache if they are located in the cache when requested, unlike LRU. FIFO has an operational cost of $O(1)$.

2.4.5 Most Recently Used (MRU)

The Most Recently Used (MRU) eviction algorithm works under the assumption that once an item has been requested, it is unlikely to be requested again, removing it when another one is added.

2.5 Icarus (ICN Simulator)

An ICN simulation environment has been created by Saino et al. [51]. The Icarus simulation environment is a publicly-available, Python-based tool for simulating caching behaviour on an ICN network. The popularity distribution of the items populating the system can be changed to reflect the desired system behaviour. Icarus enables the possibility of implementing a number of routing strategies which dictate the flow of data with the intention to populate caches in the network in the most efficient manner thus reducing the total amount of traffic on the network. Icarus also allows for any topology to be implemented. The results acquired are measurements of; average cache hit ratio, average path stretch and latency experienced on the network.

2.6 Conclusions and Scope

To conclude, the topic of video request is vast. Especially when technologies able to achieve greater network performance when applied to VoD are brought into focus. To limit the scope of this Thesis to an achievable goal, the main interests and conclusions to be taken from this brief introduction of technologies above will be as follows:

1. Variable size segmentation cannot be applied in combination with caches without a method of appropriately weighing the historic request frequency of items, thus excluding the possibility of applying the simple, and often used, cache eviction algorithms such as LRU and FIFO. For this reason the "Fixed" segmentation policy will be applied throughout this Thesis to create opportunities to cache using a more vast array of simple cache eviction algorithm. Any cache-specific research containing variable length segmentation, such as pyramid segmentation and skyscraper segmentation will be considered out of scope for this Thesis and the research contained.
2. Video popularity distributions are a point of discussion in the research community in regards to what popularity distribution, namely Zipf or Zipf-Mandelbrot, is to be considered a superior distribution for modelling video popularity data. This

Thesis will regard video popularity distributions to be a primary focus of study with special attention to the discussion surrounding video request data and the methods used to analyse which model is best to consider for the most approximate replication of real VoD data. Additional potential distributions, such as the log-normal distribution, will be considered outside of the scope of this Thesis.

3. Additionally to Video Popularity, other characteristics of video will be discussed but not analysed in the same detail as the video popularity models. This is due to the lack of available data for such analysis. However, data from the limited existing sources will be combined in an effort to provide a pseudo-realistic request simulation environment for the purpose of simulating request data to be as encompassing of all known variable characteristics as possible. This work will require leaps of judgement due to the restrictive nature surrounding the details of the characteristics involved.
4. As a method to reduce video delivery strain on a network, caching has been brought forward as a possible solution. Caching, though possible, is restricted in present day IP infrastructures, thus creating the need for alternative infrastructures to enable and support a cache-enabled network. Any-cast protocols such as CCN and ICN provide such an environment. For this reason these technologies will be brought into focus in this Thesis with a primary focus of cache implementation and development of cache eviction algorithms. A simulation environment of an ICN cache enabled network is available (Icarus [43]) and will be utilised throughout this Thesis to develop and test novel cache eviction algorithms.

3

Video Popularity Distribution Analysis

Global IP video traffic is predicted to constitute 82% of all consumer internet traffic by 2020, up from 70% in 2015 [6]. Innovation to reduce the costs of video streaming are therefore necessary and are being developed. Some examples are distributed server farms and video compression techniques, for which most solutions require simulations to stipulate just how effective one may presume specific solutions to be. Simulation data such as total video request distributions may for some VoD hosting platforms be accessible, however this is not so for most. Identifying key characteristics will open up the ability to simulate and reconstruct video request behaviour to more people, creating further opportunity for innovation. The goal of this Chapter is to identify the video request distribution of a VoD system for the purpose of reconstruction. To achieve this, two models are analysed to conclude that a specific model, with the required parameters, would be the best suited model to closely recreate the empirical dataset.

3.1 Introduction

Accurate modelling of video consumption patterns is a key factor in providing efficient utilisation of network resource. However, the lack of publicly available consumption data has meant that video consumption models have not been put under scrutiny and, instead have assumed to follow the same model, namely Zipf, as many other Internet services have; such as the World Wide Web, news feeds and email [26, 52]. Existing research has provided extensive testing that shows Zipf-like distribution to be sufficiently accurate for modelling such services [26, 52, 53]. However, the same cannot be said for video consumption models. In fact, various observations have shown that Zipf-like distributions may not be the best model for describing the popularity distribution of video consumption patterns, hence, Zipf-Mandelbrot may be a better fit.

The danger of simply assuming a Zipf-like distribution for video consumption, without supporting evidence, is that the resultant model may not be sufficiently representative of the user demands. Assuming a badly fitting model may lead to sub-optimal design choices in areas such as service planning/provisioning, utilization of network resources, accommodation of Service Level Agreement (SLA)s. A video caching system is one example where accurate video consumption models have significant impact on the network performance [14, 54, 55]. Such systems need to accommodate different types of video services, such as: VoD, Live streaming and UGC which in some cases follow vastly varying distribution models.

VoD has been argued to follow a Zipf-like consumption pattern [9, 10, 12]. However, the observed distribution of data shows a curvature on the log-log scale, which suggests a possible alternative model, namely Zipf-Mandelbrot [56]. Similar observation can be made for UGC consumption patterns [17]. This deviation may increase in the future of the Internet, with the rapid growth of user demands, the widening variety of the offered video services and the expected innovation in Internet architectures, such as

ICN [42, 50, 57].

This Chapter addresses the shortage in evaluation of video-focused consumption models. While there heve been hints that Zipf-Mandelbrot is a good model, this work shows – through empirical data – that consumption patterns do indeed comply to this distribution. This is achieved by analysing an example large-scale empirical dataset, provided by BT , as well as using synthetically generated consumption data, following Zipf and Zipf-Mandelbrot distributions. These models include standard testing methods such as; Pearson chi-square, Pearson's correlation-coefficient, as well KL divergence. Our study demonstrates that Zipf-Mandelbrot better fits realistic consumption data than the previously widely used model: The Zipf-like distribution. Furthermore, this study shows that the expected behaviour of cache-enabled video delivery systems is closer to that of the empirical dataset when using the Zipf-Mandelbrot model than the Zipf-like model.

3.2 Video Popularity Metrics

VoD systems vary in a number of ways. A summary that highlights the diversity of VoD systems previously investigated can be found in Table 3.1. Table 3.1 is comprised of a number of VoD system characteristics described in a number of publications. The years of observation, user counts, requests per day, requests per user, locations of observation and the number of unique videos are the gross level of features when expressing the diversity of VoD systems currently live. It may be assumed that the diversity of VoD systems impacts the Video Popularity Distribution, however, evidence shows that although the individual parameters may vary from one system to another, there is a commonality in the type of distribution. A good example is that presented by Cha et al. [17] in the case of Netflix [32]. The Netflix search algorithm had a sampling bias that impacted the popularity distribution, however, inspection [17] of the popularity distribution would suggest a Zipf-Mandelbrot distribution is present. When inspection of all popularity distributions of systems listen in Table 3.1 it appears that the popularity

References	Systems	Users	Days	Requests	Unique Videos/Segments
[14]	System 1	*	870	666074	2999
	System 2	*	630	14489	412
[10]		150000	219	21498338	7036
[9]	North	24059	21	1159676	536616
	South	9762	28	615166	336257
[22]		20000	61	25000	1200
[9]	BIBS	*	110	$\approx 6 * 10^4$	1506
	eTeach	280	32	$\approx 2 * 10^4$	73
BT	TV catch-up	$\approx 10^5$	7	$\approx 3 * 10^5$	≈ 2500
BT	VoD	$\approx 10^5$	7	$\approx 3 * 10^5$	≈ 11000

Table 3.1: Published statistics from VoD and UGC systems + BT provided statistics

distributions, though not the same in each example, show clear trends despite the diversity of each system and its observation. Here, a clear distinction was made between a number of consumption systems: linear, which broadcast video services; *TV-catchup*, a form of VoD where content is typically time-limited and introduced in conjunction with *linear TV*; and, *VoD-time-unlimited (VoDu)*, which includes systems such as Netflix, where a catalogue is maintained over a relatively long time.

The BT data-sets are compared and contrasted in Table 3.1 against a range of other VoD systems which all have been observed between 2000 and 2012 with a variety of properties.

3.2.1 Existing Popularity Distribution Models

Currently, there is no agreement on one popularity distribution model that would fit all video delivery systems; however, the general consensus appears to be that content, regardless of type, system and time of observation, still follows a Zipf-like distribution [9, 10].

A number of studies [12, 16, 17] have expressed caution assuming that the popularity distribution of video content is Zipf-like, due to the heavy tail and the flat head seen on a log-log scale. Cha et al. [17] suggest that perhaps the truncation of the heavy tail is caused by the system implementation, which does not make niche content easily

accessible. However, analysis of other studies [9, 10, 14, 18, 22] shows that the truncated tail, and the flat head, appear in many other VoD systems.

Alternatively, Tang et al. [14] suggests that Zipf-Mandelbrot represents VoD popularity distributions more accurately, which have also been witnessed in comparable systems, such as P2P [58]. This trend may not be exclusive to video content; but rather general multimedia-based content [16]; thereby, showing Zipf-Mandelbrot to be emerging as an alternative distribution model for describing general multimedia-based content (including video), as opposite to the conventional Zipf model.

3.2.1.1 The Pareto Principle

The *Pareto principle*, not to be confused with the *Pareto Law*, is used to describe a scenario were 20% of the categories in a probability distribution hold 80% of the chance of occurrence. The *Pareto Principle* has been observed a number of times to be present in VoD video request data [10,17,33] however, the significance of this finding is debatable and may be seen as anecdotal. For the purpose of completeness, the *Pareto Principle* will be addressed in Chapter 3.4.1.1 in relation with the BT provided empirical datasets.

3.2.2 Candidate Models for Popularity

The two models considered for replication of the user request data are Zipf [59] and Zipf-Mandelbrot [56]. The Zipf distribution is a discrete power-law distribution first used in linguistics to model the frequency of used words. Zipf-Mandelbrot is Zipf with an extra variable which creates the flat head section of the distribution and also was first applied to research and modelling in linguistics.

$$P(i) = \frac{C}{(i+V)^{\alpha}} \tag{3.1}$$

$$C = 1/\sum_{j=1}^{N} \frac{1}{j+V}^{\alpha} \tag{3.2}$$

The Zipf and Zipf-Mandelbrot distributions are modelled using Equation 3.2 and Equation 3.1. α determines the skew of the Zipf/Zipf-Mandelbrot distribution. ν determines the flatness at the top of the distribution and thus the curve of the distribution and is often called the Plateau factor. When ν is equal to 0 and $\alpha = 1$, the distribution obeys Zipf's law. The normalization fraction, C, ensures that the probability mass function $P(i)$ sums to unity over the range $[1, N]$. Strictly a Zipf distribution is where $\nu = 0$ and $\alpha = 1$. If $\alpha \neq 1$ and $\nu = 0$ $P(i)$ may be termed as Zipf-like; however, here, for brevity $P(i)$ will be simply called a Zipf distribution. $P(i)$ does not converge for $\alpha \leq 1$, however, with a finite number of items $P(i)$ is determinable, but we note that with smaller α , the tail of the distribution becomes more important.
Figures 3.2 and 3.3 demonstrate the flatness of the top of the distribution of the real user VoD and TV catch-up request data on a log-log scale for each item in the online video delivery system.

3.2.3 Fitting Models and Problem Description

Currently, there is no agreement on one popularity distribution model to fit all video delivery systems. In the following section two methods are suggested for fitting a model to an empirical video data-set, each with strengths and weaknesses. The methods suggested for fitting use KL and PCS respectively to find a sufficiently good fit to the presented empirical data. Additionally to the two methods suggested for determining likeness between the empirical data and the suggested model, a correlation measuring method called the Pearson Correlation Coefficient is used to confirm correlation between the empirical data-sets and the suggested best fitting models. The strengths and weaknesses of all methods of testing mentioned are discussed in the following Chapters.

Let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$ be the probability of occurrence of N items following a known, discrete, probability distribution model; and, let $\Phi(\ldots) = \{\phi_1, \phi_2, \ldots, \phi_N\}$ be the probability of occurrences of the same N items but according to a parametrized probability distribution model that is used to approximate Θ. The elements of Φ and Θ are strictly ordered from the highest probability to the lowest and the position of each

item is termed its rank (i.e. $1, \ldots, N$). This Thesis will be concerned with measuring the difference between the distributions and determining what it means in terms of applying video consumption patterns to systems such as caching algorithms. Choosing an appropriate distribution model, Θ, can be a notoriously difficult task, however, once a suitable distribution model has been selected Φ may provide a sufficiently similar substitute for Θ.

In addition to using KL as a method to measure "goodness-of-fit", KL presents a method by which the maximum cache miss can be estimated assuming a clairvoyant cache eviction algorithm is adopted. The method will give additional evidence, beside the traditional KL divergence, a measurement of behavioural prediction of a simulation environment where a video item popularity distribution affects the outcome which, in this implementation, is video caching miss/hit ratio.

3.2.3.1 Pearson Chi-Squared testing model

The standard use of the Pearson Chi-Squared test is used to determine if two sets of observed data are likely to originate from the same source by chance. The "goodness-of-fit" is expressed by a *p-value* produced by the test which determines the probability of the two sets originating from the same source. Here it will be used to find the closest fitting model, as illustrated in Figure 3.1, which utilizes the Pearson Chi-Squared test to quantify the "goodness-of-fit" of each model (i.e. Zipf and Zipf-Mandelbrot (Φ)) in describing an example empirical dataset (Θ) in an unconventional manner. The source of the distributions, Zipf and Zipf-Mandelbrot, is not the same as the empirical dataset, thus the Pearson Chi-Squared test, in its original form, will determine the two sets to be different (produce a p-value that is significantly low) if the datasets drawn from each probability distribution are of significantly large size. The key lies in the tests' manner of dealing with a small data-set, as it will then produce a large p-value for two sets, concluding the two sets may originate from the same source due to a limited amount of data.

An example case would be in the following two scenarios; first, one may test to see

if on a non-biased flip, a coin lands face up 49% of the time; secondly, a test to see if a during a non-biased flip, a coin lands face up 20% of the time. As one may conclude, it is more likely that the first test is harder to disprove, as it requires a significantly higher quantity of coin flips if you want to conclude beyond a 95% certainty that this statement incorrect. This is also reflected in the Pearson Chi-Squared test, as the second test would produce a *p-value* of less than 0.05 with fewer flips than the test which sees if a non-bias coin flip lands face up 49% of the time. For this reason, it may be concluded that a distribution of $49\% - 51\%$ shares a greater "likeness" to a $50\% - 50\%$ coin flip than a $20\% - 80\%$ coin flip. This method of finding the distribution with the greatest "likeness" re-purposes the Pearson Chi-Squared test to relatively compare datasets to an empirical dataset. This will become evident in the results in Section 3.4.2.1.

To determine if a single model is a superior model to reproduce an empirical probability distribution Θ the *Pearson Chi-Squared*, is used. From each model, a large quantity of Probability Density Function (PDF)s Φ are produced with a range of parameters specifically chosen for their resemblance to the empirical PDF Θ. From each of the PDFs Φ a number of datasets are produced. The datasets are a number of occurrences, each category listed in the PDFs, from which they are generated increasing in quantity. The datasets are directly compared to Θ using the *Pearson Chi-Squared* test to produce a *p-value*. As the generated datasets submitted to the *Pearson Chi-Squared* test from a specific PDF Φ increase in size the *p-values*, resulting from the test will decrease, deducing a gradually less likely probability of each of the datasets originating from the same source. Once the *p-value* drops below a previously specified value, the size of the dataset produced from a single PDF Φ is recorded as the representative "goodness-of-fit" value of that PDF Φ, before moving onto the next selected PDF Φ and repeating the process. Once all the selected PDFs Φ from all the specified models have been tested, the best "goodness-of-fit" can be deduced. Each dataset generated from a specific PDF may vary slightly, which is why for each size of dataset, the test was repeated a number of times and an average *p-value* was recorded as the representative of that test. The above mentioned process is more clearly described in Figure 3.1.

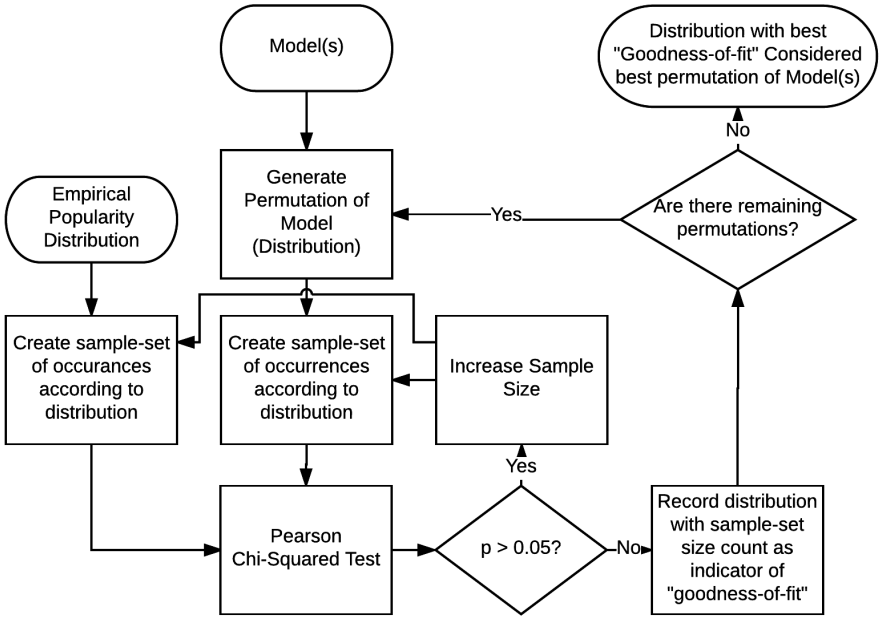


Figure 3.1: Model for performing the Pearson Chi-Squared Testing for deciding a superior fitting Model and permutation of Model.

$E = NormalisedEmpiricalDistribution$
for $\delta \in PermutationsofModel$ **do**
    $p = 0$
    $largestM = 0$
    $M = 100$
    while $p < 0.05$ **do**
        $M = M + 10$
        for $i \in 1 : 100$ **do**
            $Eset = sample(M, E)$
            $Zset = sample(M, D)$
            $p = p + PCS(Zset, Eset)$
        end for
        $p = p \div 100$
        if $M > largestM$ **then**
            $BestDist = \delta$
        end if
    end while
end for

The tests were constructed in a manner that would repeat the Pearson Chi-squared test incrementally increasing the number of sample requests used to generate the data-sets compared. To avoid false positives, forty data-sets were generated with this method and subsequently tested. The average was calculated to represent the p-value used to consider the "goodness-of-fit". The data-set of the models that has $p > 0.05$ for a larger quantity of samples in their data-sets is considered to have a greater likeness to the empirical data.

The final test enters a single permutation (distribution) of each respective model that was found to be the best representative of their respective model and compares the numbers of samples for which the $p > 0.05$ remained for the largest set of samples. The results of this final comparison present a winning model which could be considered a superior model through which to generate a pseudo-realistic VoD popularity distribution.

note: The p value chosen for the Pearson Chi-Squared test (0.05 in the case above) is arbitrary. It can be changed and is to not influential as to which distribution is found to hold the best "goodness-of-fit". p is preferred to be of a low value as the purpose of the test is to disprove the null hypothesis, the high confidence in disproving the null hypothesis with a $p < 0.05$ can be considered to be more in-line with the original purpose of the test.

3.2.3.2 Pearson Correlation Coefficient

The Pearson Correlation Coefficient is used to confirm the results deduced by the tests performed using KL and PCS with a measurement of the correlation between the respective distributions found to hold the greatest "goodness-of-fit" measurements. The Pearson Correlation Coefficient cannot be used to decide a distribution model to have great likeness but can assist in confirming correlation between two separate distributions. The resulting distributions of the KL and PCS tests will be presented in a Q - Q plot against the empirical data to represent a visual representation for the correlation PCC confirmed.

3.2.3.3 Kullback-Leibler Divergence

A common metric of the difference between two distributions is the Kullback-Liebler divergence, $I(\Theta\|\Phi)$, [60] which is defined as:

$$I(\Theta\|\Phi) = \sum_{i=1}^{N} \Phi_i \log(\Phi_i/\Theta_i) \tag{3.3}$$

I can be interpreted in a number of ways. This Thesis will interpret it as the measure of *surprisal* [61], measured in bits. Specifically, the surprisal, I, is the log-odd ratio that the *a priori* belief of a process thought to obey Φ is actually found to obey the *a posterior* distribution Θ. I can be measured in bits, *nats* or other units depending on the choice of logarithm, in this Thesis, the natural logarithm will be used as it avoids adjustment terms later on.

In other words, the value given by KL divergence is the logarithmic difference between the probabilities Θ and Φ, where starting probabilities are presumed Θ.

3.2.4 Cache Miss Error Ratio Calculation using KL

Additionally to finding the model to which empirical video request date coheres it is considered important to see if, outside of standard empirical measurement techniques, behaviours expected from video simulation environments testing cache eviction policy performance reflects the findings of the empirical measurements. A simulator, such as the one suggested, is found in Chapter 3.4.3. Additionally to this simulator, an empirical measurement of probability of cache miss ratio is introduced to further substantiate the findings using KL.

Singularly, KL can aid in determining the cache miss error if applied correctly. Assuming KL to be defined as in Section 3.2.3.3 the divergence I is a result from information theory that can be related to probability theory through an inequality termed Pinsker's inequality, which was later refined independently by Csiszár, Kemperman, and Kullback [62]:

$$V(\Theta, \Phi) \leq 2I(\Theta\|\Phi) \tag{3.4}$$

where $V(\Theta, \Phi)$ is the *total variation* between Θ and Φ. For a discrete probability distribution with finite support over N-items:

$$V(\Theta, \Phi)^2 = \sum_{i=1}^{N} |\theta_i - \phi_i| \tag{3.5}$$

It can now be seen how the divergence I between a parametrized approximation Φ, for an actual distribution Θ, can be used to predict useful results when applied to caching; for example the error in the cache-hit ratio when analysing a caching system for video distribution. Before deriving the result that determines the error, the following lemma which considers a discrete probability distribution $P = \{p_1, \ldots, p_N\}$ where events are combined as a single event to form a new probability distribution denoted as $P_{\{\alpha,\Omega\}} = \{p_1, \ldots \{p_\alpha + \ldots + p_\Omega\}, \ldots p_N\}$ is needed. Note that α and Ω may be selected from any point in P and that multiple ranges of events could be combined and denoted as $P_{\{\alpha,\kappa\}\{\rho,\Gamma\}}$.

Lemma 1. $I(\Theta_{\{\alpha,\Omega\}} \| \Phi_{\{\alpha,\Omega\}}) \leq I(\Theta \| \Phi)$

Proof. The proof will use Jensen's inequality [63]:

$$f\left(\sum_i a_i x_i\right) \leq \sum_i a_i f(x_i)$$

where the convex function is defined as $f(x) = x \log(x)$ and $\sum_i a_i = \sum_{i \in \{\alpha \ldots \Omega\}} \frac{\Phi_i}{\phi} = 1$ if $\varphi = \sum_{i \in \{\alpha \ldots \Omega\}} \phi_i$ such that Jensen's inequality holds. By setting $x_i = \frac{\theta_i}{\phi_i}$ it can be found that

$$f\left(\sum_{i \in \{\alpha \ldots \Omega\}} \frac{\phi_i}{\varphi} \frac{\theta_i}{\phi_i}\right) \leq \sum_{i \in \{\alpha \ldots \Omega\}} \frac{\phi_i}{\varphi} f\left(\frac{\theta_i}{\phi_i}\right)$$

Dropping the limits $\{\alpha \ldots \Omega\}$ from summations for brevity, substituting for the function f and using $\sum_{i \in \{\alpha \ldots \Omega\}} \theta_i = \vartheta$ gives

$$\sum_i \frac{\phi_j}{\varphi}\frac{\theta_j}{\phi_j}\log\left(\sum_j \frac{\phi_j}{\varphi}\frac{\theta_j}{\phi_j}\right) \leq \sum_i \frac{\phi_i}{\varphi}\frac{\theta_i}{\phi_i}\log\left(\frac{\theta_i}{\phi_i}\right)$$
$$\vartheta \log\frac{\vartheta}{\varphi} \leq \sum_i \theta_i \log\left(\frac{\theta_i}{\phi_i}\right)$$

Noting that the two sides of the inequality contribute to the following formulations of the respective Kullback-Leibler divergences

$$I(\Theta\|\Phi) = \sum_{i\in\{\alpha\ldots\Omega\}} \theta_i \log\left(\frac{\theta_i}{\phi_i}\right) + \sum_{i\in\{1\ldots N \mid i\notin\{\alpha\ldots\Omega\}\}} \theta_i \log\left(\frac{\theta_i}{\phi_i}\right)$$

and

$$I(\Theta_{\{\alpha,\Omega\}}\|\Phi_{\{\alpha,\Omega\}}) = \vartheta \log\frac{\vartheta}{\varphi} + \sum_{i\in\{1\ldots N \mid i\notin\{\alpha\ldots\Omega\}\}} \theta_i \log\left(\frac{\theta_i}{\phi_i}\right)$$

completes the proof. □

Consider an idealised cache that holds the top C items and when Φ is invariant.

Definition 1. *The actual cache hit ratio from the system obeying the distribution* Φ *is* $h = \sum_{i=1}^{C} \phi_i$ *whereas the estimated ratio according to the approximated distribution* Θ *is* $h' = \sum_{i=1}^{C} \theta_i$.

The following lemma can then be constructed.

Lemma 2. *The absolute error in the estimate of the cache hit ratio* $|h - h'|$ *is bound by the Kullback-Leibler divergence with the relationship:*

$$\left|h - h'\right| \leq \sqrt{\tfrac{1}{2} I(\Theta\|\Phi)}$$

Proof. The proof of the Lemma follows directly from Pinsker's inequality [62] for two

discrete distributions $P = \{p_i \dots p_N\}$ and $Q = \{q_i \dots q_N\}$ each with the same finite support over N items:

$$\left(\sum_{i=1}^{N} |p_i - q_i|\right)^2 \leq 2I(P\|Q)$$

Consider that the actual distributions of cache-hit/cache-miss is given by the two value distribution $\Theta_{\{1\dots C\}\{C+1\dots N\}}$ and likewise the estimated distribution is $\Phi_{\{1\dots C\}\{C+1\dots N\}}$ according to the earlier defined notation such that $h = \Theta_{\{1\dots C\}}$ and $h' = \Phi_{\{1\dots C\}}$. Applying Pinsker's inequality above gives:

$$\left(\left|h - h'\right| + \left|(1-h) - (1-h')\right|\right)^2 \leq I(\Theta_{\{1\dots C\}\{C+1\dots N\}}\|\Phi_{\{1\dots C\}\{C+1\dots N\}})$$

Applying Lemma 1 and rearranging the left hand side gives:

$$\left(2\left|h - h'\right|\right)^2 \leq I(\Theta_{\{1\dots C\}\{C+1\dots N\}}\|\Phi_{\{1\dots C\}\{C+1\dots N\}}) \leq I(\Theta\|\Phi)$$

and the proof is complete. □

3.3 Application in Video Delivery Systems

Example environments for which knowledge of the popularity distribution for a video delivery system is useful are plentiful. Any simulation of video delivery may choose to apply the video popularity distribution when testing any number of properties such as: capacity of links or servers, load balancing, HTTP or DNS redirection [64,65] and many more.

One area where the popularity distribution of the input data influences the performance of a system is in cache enabled distributed networks. Content Centric Networking (CCN) [57] and Information Centric Networking (ICN) [44] provide us with protocols on which caching is possible and on which we can implement a Video Delivery Systems. The CCN and ICN protocols can be simulated to show caching metrics to reveal differences in performance under a variation of circumstances, thus providing a perfect way to demonstrate the behaviour of the system with input data following different popularity

distributions.

The motivation for identifying the best method of generating a realistic request frequency distribution for video delivery systems is to enable the possibility to achieve accurate system request behaviour in video request simulation/modelling. One example is to simulate the reduction of traffic on a network through the implementation of local caching on, for example, an Information Centric Networking (ICN) network such for the simulator named "Icarus" developed by Saino et al. [51]. Cache Eviction algorithms may perform differently when subjected to a Zipf-like request frequency distribution instead of Zipf-Mandelbrot distribution (both modelled to closely resemble the original request distribution) and thus could possibly identify inaccurate results should one distribution be selected for simulation over another.

3.4 Tests and Evaluation

3.4.1 Empirical Data

The BT provided popularity distributions shown in Figures 3.2 and 3.3 are observed request distributions which are labelled as "real". A week of observations make up the TV catch-up and VoD data. There are two types of video delivery systems as shown. The two unique systems provide a contrast in request behaviour that will aid in confirming if video request distributions can be accurately modelled and to what extent modelling of video request distributions can be considered accurate.

The TV catch-up and VoD systems have key differences in the method they function. Some of these differences are: Lifetime of videos - The TV catch-up system holds videos for thirty days and was monitored for the duration of seven days. Every 24 hours a 30th of the total amount of videos is gradually removed and an equal amount of videos are introduced back into the system. The VoD system holds videos for an undetermined amount of time. Method of selection - TV catch-up videos are also selected on the grounds that they were previously broadcast on TV. This entails that some items only hold relevance for a day (e.g. news items) and others may remain popular throughout

the week (e.g. popular TV shows). All items remain on the system all thirty days regardless of their type. VoD holds specifically videos that are likely to be requested frequently. The type of videos requested would hold a more constant request rate which is not dependent on their time of entry into the system.

3.4.1.1 Pareto Principle

The Pareto principle is often quoted with respect to popularity of video watching patterns and thus is briefly analysed here for the data obtained for this study. In the case of TV catch-up, data reveals that the Pareto Principle applies as 20% of the most popular videos make up 83% of the total number of requests in the TV catch-up observations. This matches work by others [10, 14, 22] who also found that the video request matches the Pareto Principle. Cha et al. [17] considered user generated content (UGC) and Yu at al. [10] considered data which is Video on Demand (VoD). Due to the different nature of the systems in question the Pareto Principle does apply to all with small variations. The UGC VoD system in [17] is more extreme with 10% of videos receiving nearly 80% of the total number of requests. The nature of UGC means that there are more videos and therefore, in proportion, fewer popular videos. While the Pareto principle has seen widespread adoption as an approximate benchmark it is not, in itself, detailed enough to be used for accurate models. Consequently, this Thesis will look to more detailed modelling.

3.4.2 Finding Resembling Zipf & Zipf-Mandelbrot Distributions

The fitting method proposed suggests selecting multiple models with multiple parameters (Φ), to be measured directly in relation to an unknown discrete popularity distribution (Θ) to receive a *goodness-of-fit* measurement. To find the superior model to represent the unknown probability distribution a single candidate from each model is to be compared. The candidate distribution suggested from each model will represent the distribution that holds the greatest likeness (P) to said model. Zipf and Zipf-Mandelbrot are the models selected for comparison, thus many permutations of each

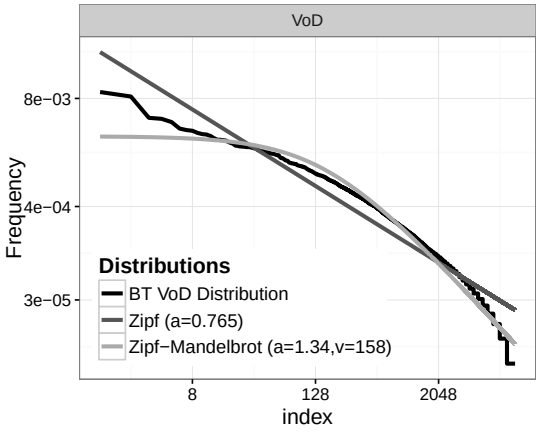


Figure 3.2: PCS Fitting results Popularity Distributions Zipf, Zipf-Mandalbrot and VoD Empirical Data-sets

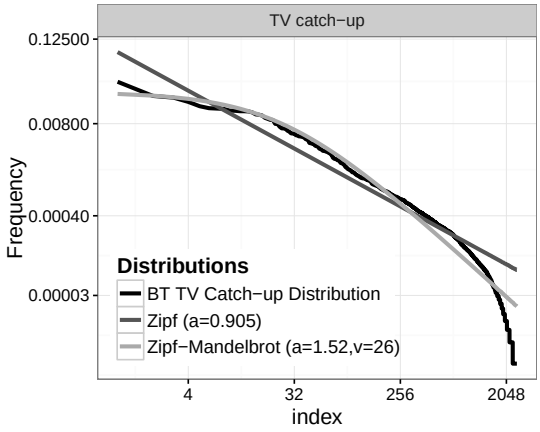


Figure 3.3: PCS Fitting results Popularity Distributions Zipf, Zipf-Mandalbrot and TV catch-up Empirical Data-sets

model are compared to the unknown discrete popularity distributions provided by BT.

Two methods of comparing are presented and are discussed in detail in Chapter 3.2.3. The methods use PCS and KL respectively.

3.4.2.1 Pearson Chi-Squared Fitting

The Pearson Chi-Squared fitting method incrementally increases the size M of sample-set θ according to a probability distribution from a known model Θ which is, with each incrementation, subjected to the Pearson Chi-Squared test in conjunction with a sample-set drawn from the unknown probability distribution Φ. The null hypothesis in the Pearson Chi-Squared test assume the tested distributions are of the same source and share a common model. If the the Pearson Chi-Squared tests returns $p < 0.05$, the null hypothesis can be rejected - indicating the two sample sets do not originate from the same probability distribution with a confidence interval of 95%. The sample-set θ for which the null hypothesis is rejected with the largest sample-size is considered the superior fit within the given model Θ.

The algorithm by which a Zipf-like and Zipf-Mandelbrot distribution can be selected to model the empirical data iteratively fits a large range of permutations of each model using the Pearson Chi-Squared test and conclude to have the greatest resemblance to the empirical data-set. The Pearson Chi-Squared test provides a measurement of the

probability of difference observed in two distributions by chance. This means increasing the number of samples taken according to each popularity distribution will provide a measurement of how likely it is that the two compared sample-sets are generated from the same probability values. As the sample-sets increase, the Pearson Chi-Squared test will increasingly consider the probability of the two sample-sets to be less likely to have originated from the same probability, thus more likely to dispel the null hypothesis. As Zipf-like and Zipf-Mandelbrot distribution will not exactly replicate the empirical data-sets, the test provides us with a method by which the distribution showing the greatest resemblance to the empirical data will dispel the null hypothesis with the largest sample-set. (The testing model can be found in Chapter 3.2.3.1)

$$\alpha = 0.5, 0.51, 0.52, 0.53, \ldots, A \tag{3.6}$$

$$\nu = 0, 1, 2, 3, \ldots, V \tag{3.7}$$

A large set of Zipf-like and Zipf-Mandelbrot distributions are generated (permutations of the models) applying a large set of α and ν variables to the Zipf-like function and Zipf-Mandelbrot function as presented in the Equations 3.6 and 3.7. The size of the data-sets, N, should be equal to the length of the empirical data-set for each individual PCS test performed. For each distribution generated the PCS test will confirm the sample-size required for which the chosen distribution of the model can no longer uphold the null hypothesis. The Zipf-like and Zipf-Mandelbrot distributions for which the null hypothesis was not rejected for the greatest sample size M are declared the best Θ, Zipf-like and Zipf-Mandelbrot in this particular representation, distributions of the empirical data.

The resulting Zipf-like and Zipf-Mandelbrot distributions are shown in Table 3.2 to illustrate the resemblance between the empirical data-set and the candidate distributions. Table 3.2 contrasts the results for the model "Zipf" to the model "Zipf-Mandelbrot". The quantity of requests required to dispel the null hypothesis for Zipf in contrast to the quantity required for Zipf-Mandelbrot is very large, thus presenting

	Model	Optimised Values	Requests
TV Catch-up	Zipf	$\alpha = 0.905$	2400
TV Catch-up	Zipf-Mandelbrot	$\alpha = 1.52, \nu = 26$	22300
VoD	Zipf	$\alpha = 0.765$	12050
VoD	Zipf-Mandelbrot	$\alpha = 1.34, \nu = 158$	76200

Table 3.2: Pearson Chi-Squared optimised result correlation between Zipf-like & Zipf-Mandelbrot

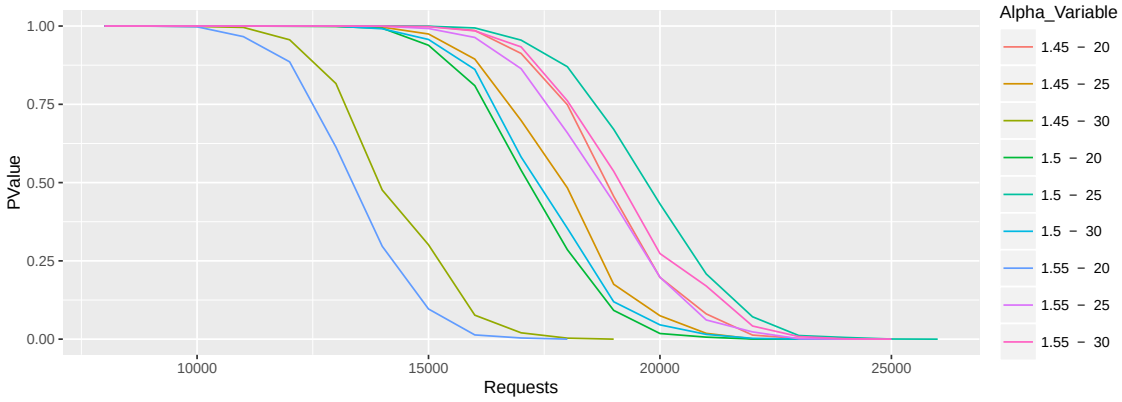


Figure 3.4: P values as occurrences increase | TV catch-up data-set (the parameter named "Variable" in this figure is a place-holder for ν)

great evidence of a superior model to which the empirical data conforms.

The p value to determine when the null hypothesis could be rejected was set at $p < 0.05$. This value can be arbitrarily chosen but more specifically was picked as $p < 0.05$ is frequently considered to a confidence level sufficiently low to reject the null hypothesis in many testing environments where the Pearson Chi-Squared test can be applied. To illustrate the insignificance of choosing a different p value where $1 > p > 0$ can be seen in Figure 3.4.

The contrasting request quantities for Zipf can be seen in Figures 3.7 and 3.5 and for Zipf-Mandelbrot in Figures 3.6 and 3.8. Each point in the graph/grid presented represents a single permutation of each respective model (Zipf & Zipf-Mandelbrot) and at which request quantity of requests the PCS fitting technique produced a p-value less than 0.05. The result from each model found to be the best suited matches to the empirical data-sets were plotted in Figures 3.2 and 3.3 to provide a visual perspective to confirm the method.

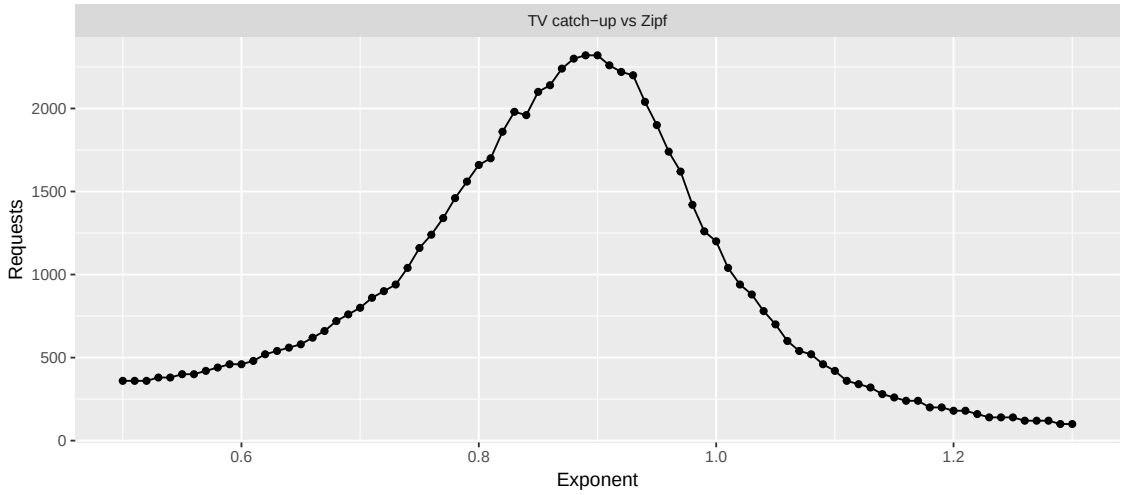


Figure 3.5: Fitting TV Catch-up to Zipf using Pearson Chi-Squared. Each point represents the point at which the p-value returned by the Pearson Chi-Squared fell below 0.05 as requests increased.

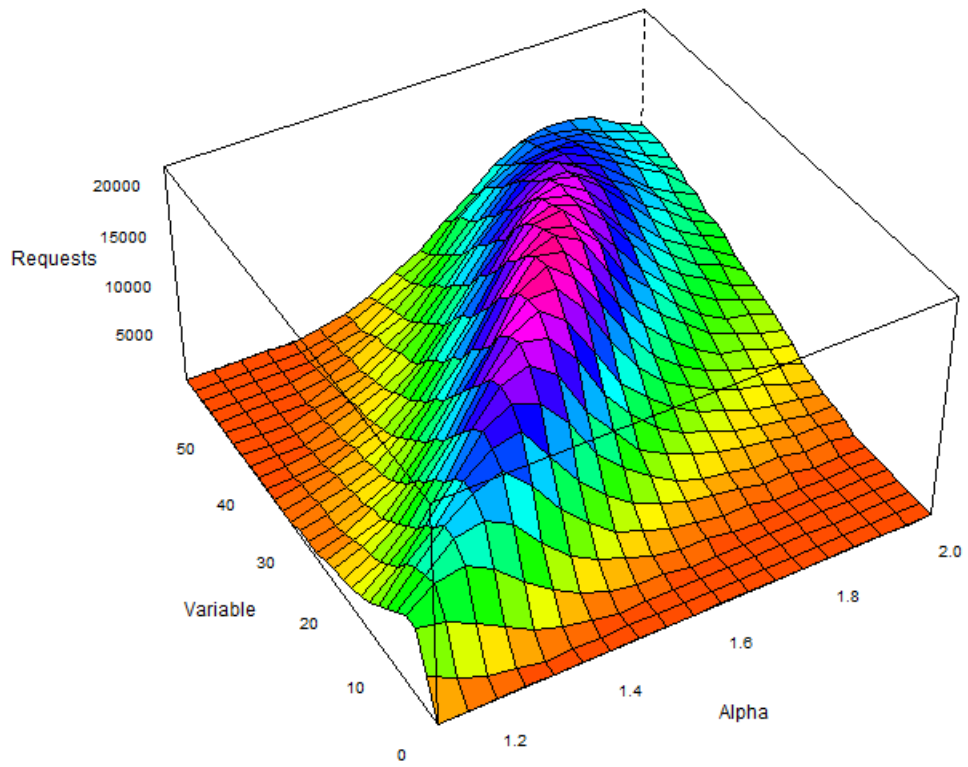


Figure 3.6: Fitting TV Catch-up to the Zipf-Mandelbrot model using Pearson Chi-Squared. Each point represents the point at which the p-value returned by the Pearson Chi-Squared fell below 0.05 as requests increased. (the parameter named "Variable" in this figure is a place-holder for ν)

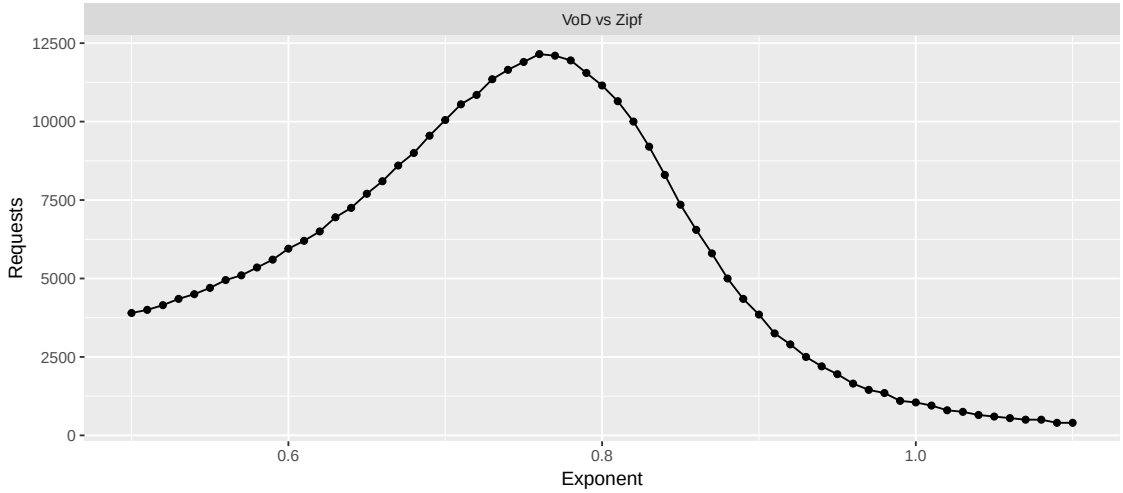


Figure 3.7: Fitting VoD to Zipf using Pearson Chi-Squared. Each point represents the point at which the p-value returned by the Pearson Chi-Squared fell below 0.05 as requests increased.

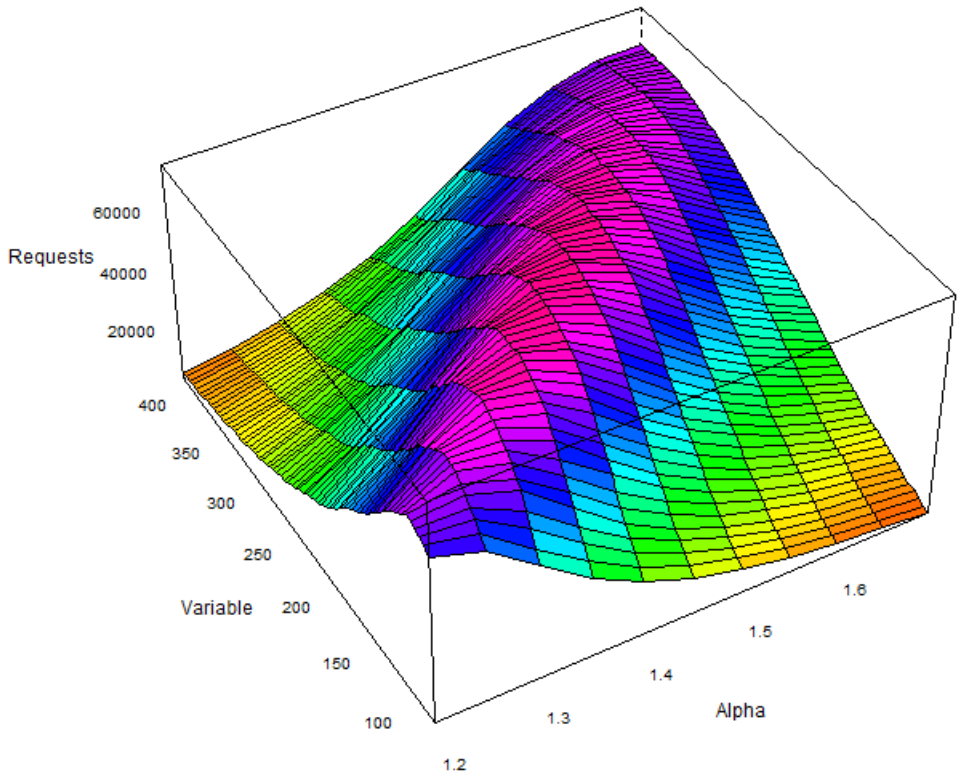


Figure 3.8: Fitting VoD to the Zipf-Mandelbrot model using Pearson Chi-Squared. Each point represents the point at which the p-value returned by the Pearson Chi-Squared fell below 0.05 as requests increased. (the parameter named "Variable" in this figure is a place-holder for ν)

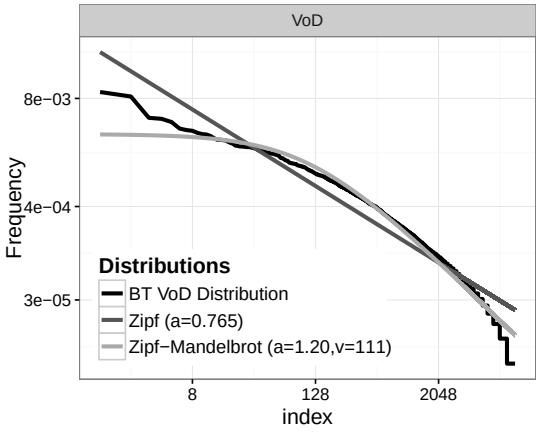


Figure 3.9: KL Fitting results including: Popularity Distributions Zipf, Zipf-Mandalbrot and VoD Empirical Data-sets

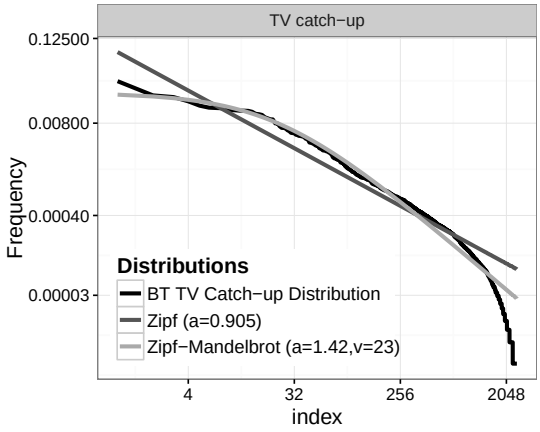


Figure 3.10: KL Fitting results including: Popularity Distributions Zipf, Zipf-Mandalbrot and TV catch-up Empirical Data-sets

A gradient ascent fitting method was initially considered for the purpose of finding the best results, however the PCS results are such that the gradient ascent process can transverse to a local maxima. For this reason a more general approach was taken by which a large range of variables were submitted to the PCS test from which the best results were recorded.

3.4.2.2 Kullback-Leibler Fitting

Kullback-Leibler (KL) provides a method of finding the expected divergence between two normalised distribution when assuming Φ represents Θ as described in Section 3.2.3.3. The error is represented in the number of bits required to code Θ assuming Φ is the information representing Θ. The measured KL values do not represent a "true metric". However, they do quantify a distance between two distributions, thus providing a relative measurement of error for two contrasting distributions (Φ) in relation to empirical data-sets(Θ). KL Divergence is presented as shown in Equation 3.8.

$$I(\Theta \| \Phi) = \sum_{i=1}^{N} \Phi_i \log(\Phi_i / \Theta_i) \tag{3.8}$$

KL is applied to many permutations of each model in question. The resulting values KL produces are so that the smallest value is equal to the smallest error, thus providing

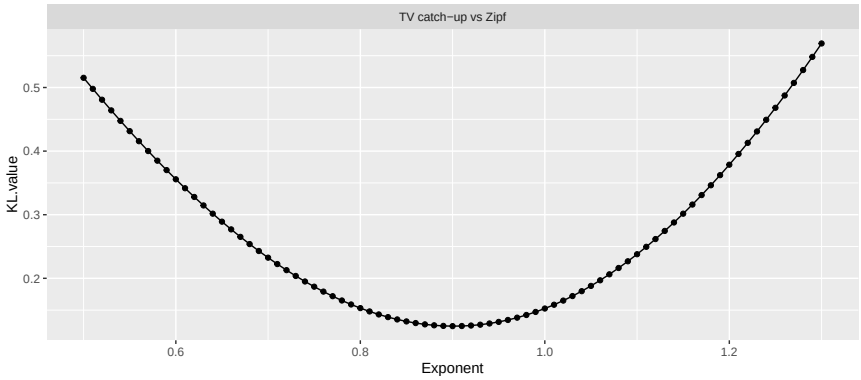


Figure 3.11: Fitting results for TV Catch-up to Zipf using Kullback-Leibler. The KullBack-Leibler value is plotted for when Θ and Φ were directly compared using a varying α variable.

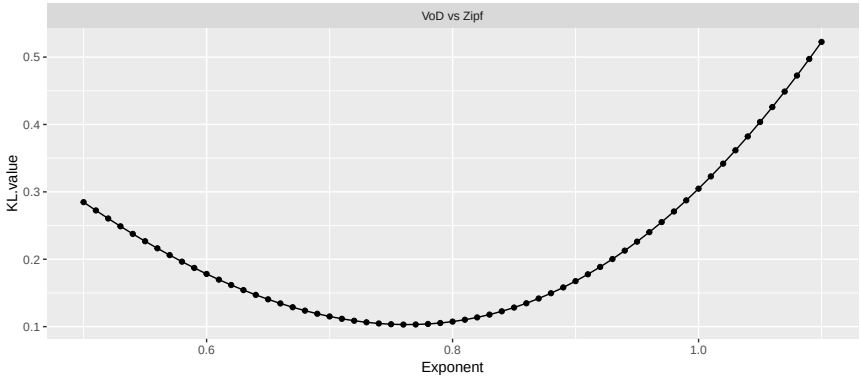


Figure 3.12: Fitting results for VoD to Zipf using Kullback-Leibler. The KullBack-Leibler value is plotted for when Θ and Φ were directly compared using a varying α variable.

the superior fit. Concluding this, the results were as illustrated in the Figure 3.11, Figure 3.12, Figure 3.13 and Figure 3.14 with each point in the figures representing an individual permutation of the model listed in the titles of the figures. The two models Zipf and Zipf-Mandelbrot were each applied to the empirical BT data-sets VoD and TV catch-up to find the best model for each set of data.

The KL results found to represent the empirical data best, to the largest degree, can be found in Table 3.3. The results indicate a preference for Zipf over Zipf-Mandelbrot considering both TV catch-up and VoD as each set produces a largely decreased KL value.

To illustrate the variance of the distributions with the lowest KL value of each respective

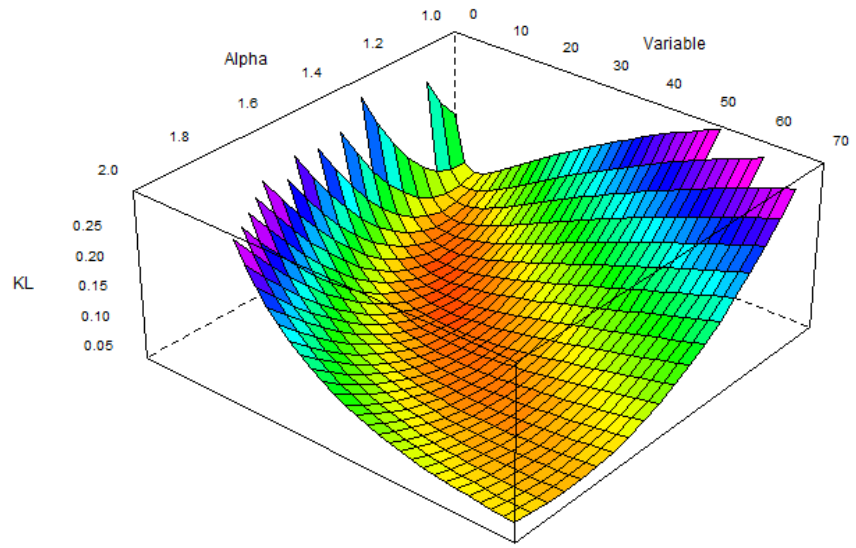


Figure 3.13: Fitting results for TV Catch-up to Zipf-Mandelbrot using Kullback-Leibler. The KullBack-Leibler value is plotted for when Θ and Φ were directly compared using varying α and ν variables (the parameter named "Variable" in this figure is a place-holder for ν).

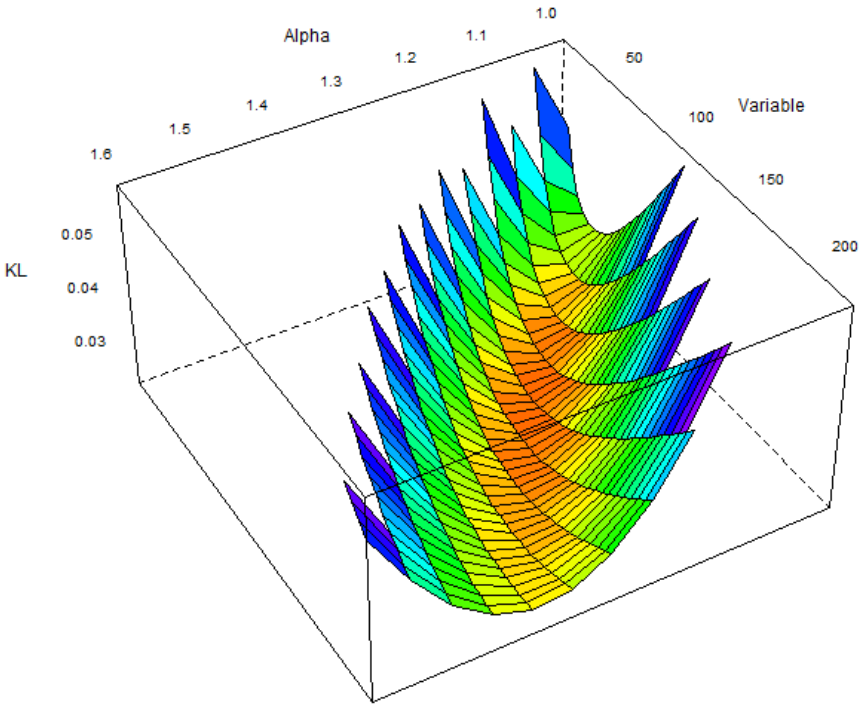


Figure 3.14: Fitting results for VoD to Zipf-Mandelbrot using Kullback-Leibler. The KullBack-Leibler value is plotted for when Θ and Φ were directly compared using varying α and ν variables (the parameter named "Variable" in this figure is a place-holder for ν).

	Model	Optimised Values	Divergence	Divergence of top 50% of content
TV Catch-up	Zipf	$\alpha = 0.900$	0.125	0.0745
TV Catch-up	Zipf-Mandelbrot	$\alpha = 1.42, var = 23$	0.0215	0.0120
VoD	Zipf	$\alpha = 0.765$	0.103	0.0610
VoD	Zipf-Mandelbrot	$\alpha = 1.20, var = 111$	0.0230	0.0166

Table 3.3: Kullback-Leiber optimised result correlation between Zipf-like & Zipf-Mandelbrot

model against TV catch-up and VoD in Figures 3.9 and 3.10. This figure shows the likeness shared between Zipf, Zipf-Mandelbrot and the empirical data-sets on a log-log scale. The measurements KL provides would suggest the Zipf and Zipf-Mandelbrot distributions are the distributions with the least amount of error if they were to be a representation of the empirical data provided (VoD and TV catch-up).

3.4.2.3 Pearson Correlation Coefficient (PCC)

The Pearson Correlation Coefficient provides a measurement of the correction between two sets of variables. The correlation expected to confirm if the Zipf-Mandelbrot or Zipf distributions correspond closely to the empirical data would be of a positive number close to 1 (a positive correlation). The distribution which most closely resembles the empirical data for both VoD and TV-catch up is considered a superior fit over its counterpart.

The Zipf and Zipf-Mandelbrot distributions subjected to the Pearson Correlation Coefficient test were those produced in the Pearson Chi-squared and KL based tests. The Pearson Chi-Squared test provides a method of producing an empirical measurement of closeness, fit for permutations within the bounds of the models provided. The KL based test provides a method similar to the PCS based test using the KL divergence results.

It is important to note the Pearson Correlation Coefficient does not provide a method of exactly matching two distributions, but instead confirming that the two distributions change in a relatively consistent and similar manner.

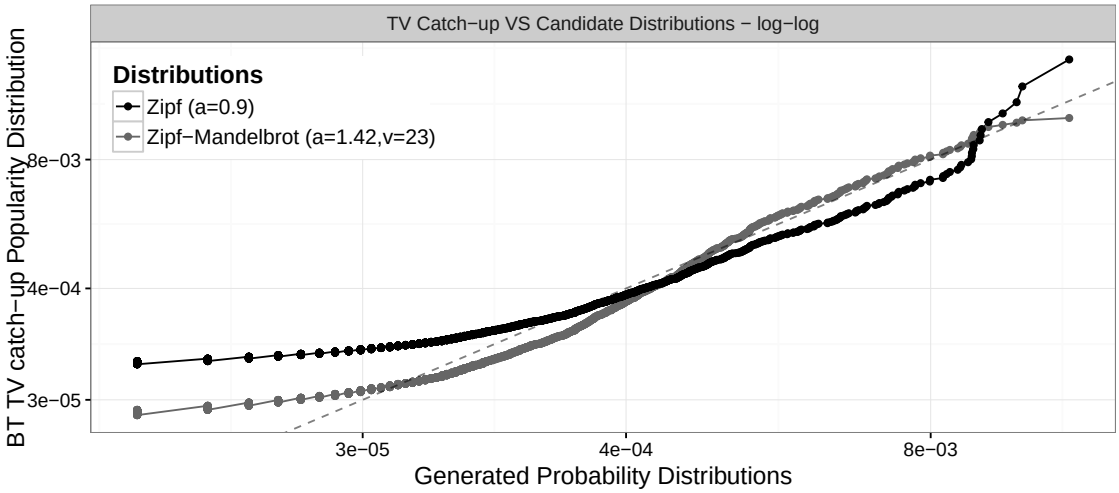


Figure 3.15: Q - Q plot including: contrasts Popularity Distributions Zipf, Zipf-Mandalbrot vs. TV catch-up Empirical Data-set

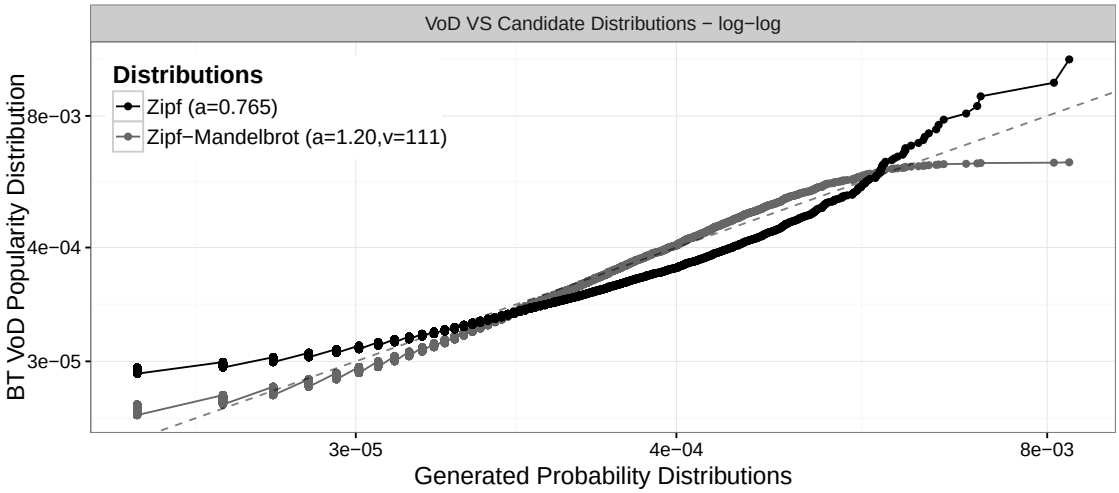


Figure 3.16: Q - Q plot including: contrasts Popularity Distributions Zipf, Zipf-Mandalbrot vs. VoD Empirical Data-set

	TV			
	Test	Correlation	K-transform	P-value
Zipf	KL	0.8575074	0.847 - 0.867	$< 2.2^{-16}$
Zipf-M	KL	0.9851084	0.984 - 0.986	$< 2.2^{-16}$
Zipf	PCS	0.8551071	0.844 - 0.865	$< 2.2^{-16}$
Zipf-M	PCS	0.9845072	0.983 0.986	$< 2.2^{-16}$

Table 3.4: TV Catch-up PCC comparison between Zipf-like & Zipf-Mandelbrot

3.4.2.4 TV Catch-Up

The correlation can be considered highly positive as values in Table 3.4 demonstrate when comparing the TV catch-up popularity distribution and the Zipf-Mandelbrot, as the Zipf distributions, in the results from the KL based analysis, as well as the results from the PCS based analysis. The P-value is used to determine a large enough test group to consider the results valid. ≈ 2500 elements represent a significantly large number of elements to validate the value produced correlation tests.

To further illustrate the likeness of the distributions, they were plotted on opposing axis (Q-Q plot) in Figures 3.16 and 3.15. A line was drawn through the centre of Figures 3.16 and 3.15 with equal X Y values to give a good correlation reference point when inspecting correlation values. The correlation is plotted on a log-log scale to illustrate the distribution of popularity. The figure illustrates that Zipf-like, though sharing similar likeness to the real data, does not match the real popularity distribution as well as the Zipf-Mandelbrot representation which is particularly obvious when inspecting the tail of each figure. Another apparent difference is the difference of the most popular items between the Zipf-like distribution and the real data. The Zipf-Mandelbrot has a much greater likeness when considering popular items.

3.4.2.5 Video On Demand

The correlation can be considered highly positive as values in Table 3.4 demonstrate when comparing the VoD popularity distribution and the Zipf-Mandelbrot, as the Zipf distributions, in the results from the KL based analysis, as well as the results from the

	VoD			
	Test	Correlation	K-transform	P-value
Zipf	KL	0.8645955	0.860 - 0.869	$< 2.2^{-16}$
Zipf-M	KL	0.9334864	0.931 - 0.935	$< 2.2^{-16}$
Zipf	PCS	0.8645955	0.863 - 0.872	$< 2.2^{-16}$
Zipf-M	PCS	0.9243367	0.922 - 0.927	$< 2.2^{-16}$

Table 3.5: VoD PCC comparison between Zipf-like & Zipf-Mandelbrot

PCS based analysis. The P-value is used to determine if a large enough test group is used to consider the results valid. VoD contains $\approx$ 11000 elements which is a significantly large number to validate the correlation test.

Figure 3.16 illustrates a log-log Q-Q plot representing the correlation between the VoD generated Zipf and Zipf-Mandelbrot data-set and the Empirical data-set. A perfect correlation would be identifiable if the two quantiles were identical (represented with a straight line between the y and x axis). Figure 3.16 has been included to show that Zipf, even though it has similar likeness to the real data, does not match the real popularity distribution as well as the Zipf-Mandelbrot representation. The most notable difference when comparing Zipf and Zipf-Mandelbrot against the real VoD data is that Zipf-Mandelbrot contains a great likeness in the main body of the distribution. When plotted on a Q-Q log-log scale, the most frequently requested items in the Zipf distribution are much unlike the real data. The Zipf-Mandelbrot distribution achieves a much greater likeness in the most frequently requested portion with the exception of the two items most frequently requested items.

3.4.2.6 PCC Evaluation

The PCC test provides a metric to suggest if two PDFs share a high correlation, suggesting the two data-sets change in a similar manner. PCC is not designed to confirm if two sets of data originate from the same source, however it does show if the sources are related. This means if a strong correlation can be considered present, a greater likeness

can also be considered in combination with other tests.

The main interest in the PCC test is to detect if there is a greater correlation to be obtained between two models. The models for consideration are Zipf-Mandelbrot and Zipf. To detect a correlation PCC has been applied to the best results from each model (Zipf and Zipf-Mandelbrot) from the PCS and KL based tests performed in Chapters 3.4.2.1 and 3.4.2.2.

The results from the TV catch-up and VoD PCC tests are in Table 3.4 and 3.5 from which the conclusion can be drawn that the Zipf-Mandelbrot model shares a greater correlation to the empirical data, for the case of TV catch-up and VoD, than Zipf by a factor of ≈ 1.1. An increased correlation of ≈ 0.1 can indeed be considered significant and thus provides an indication as to which model shares greater likeness to the empirical data.

3.4.3 Network Caching Simulation Environment (Icarus)

The video popularity distribution of a system can have detrimental impact on a number of performance aspects of a CDN. Poor planning using an incorrect video popularity distribution can lead to sub-optimal design choices in areas, such as service planning/provisioning, utilization of network resources, accommodation of SLAs. One area where using an incorrect model to imitate video request behaviour can have adverse effects is cache-enabled networks such as those that can be set-up and launched using ICN. To achieve a measurement of the variance in a cache enabled system between empirical data and a theorised model thought to imitate the empirical data would provide a method of confirming likeness.

One ICN simulation environment has been created by Saino et al. [51]. The Icarus simulation environment is a publicly-available, Python-based tool for simulating caching behaviour on an ICN network. The popularity distribution of the items populating the system can be changed to reflect the desired system behaviour. Icarus enables the possibility of implementing a number of routing strategies which dictate the flow of data with the intention to populate caches in the network in the most efficient manner thus reduc-

ing the total amount of traffic on the network. Icarus also allows for any topology to be implemented. The network topology used for the simulations is GEANT [66] (European academic network). More topologies were initially submitted; however topologies GARR [67], GEANT [66], TISCALI [68] and WIDE [69] produced insignificantly different results and thus were omitted. The results acquired are measurements of; average cache hit ratio, average path stretch and latency experienced on the network. All of the acquired measurements were compared to point out differences in behaviour for the submitted video request distributions. A number of popularity metrics will be subjected to testing with regards to request behaviour however, the simulations will not include details such as; item popularity decay over time, item removal over time and introduction of new items. Instead a static workload will be subjected for testing.

The method to predicting the cache miss ratio proposed in Chapter 3.2.4 will be additionally shown side-by-side to the Icarus results as to further substantiate the correlation between the finding of the empirical measurements using PCS and KL in relations to which model may be most closely associated to the empirical data-sets.

3.4.3.1 KL Predicted Cache Miss Ratio

As described in Chapter 3.2.4 it is achievable to estimate the cache-miss bound relative to the original data when submitting the generated probability distribution, as well as the original empirical data, to the KL divergence test.

The absolute error in the estimate of the cache hit ratio $|h - h'|$ is bounded by the Kullback-Leibler divergence with the relationship:

$$\left|h - h'\right| \leq \sqrt{\tfrac{1}{2} I(\Theta \| \Phi)}$$

The results for the VoD and TV catch-up empirical data-sets can be found in Table 3.3. As the Icarus experiments will not experience cache sizes C to increase beyond 50% of the size of total items N, the normalised top 50% of the empirical data to the top 50% of the generated probability distribution can be applied to achieve a more narrow bound cache error prediction.

Note: The results assume a clairvoyant cache. This is not simulated in using the Icarus environment, thus assuming strict boundaries would be incorrect.

3.4.3.2 Expected results beyond KL

It can be speculated that the Zipf-like distribution receives a lower mean hit-rate and a higher average path stretch due to fewer popular items having an elevated amount of requests when compared to the Zipf-Mandelbrot distribution which sees fewer requests towards the tail section of the Zipf-like curve but more requests towards the middle of the distribution. This, it could be argued, may not hold true for when the cache size is small. Zipf sees the most popular items receive a tremendous amount of requests where Zipf-Mandelbrot has a number of items that are the most popular, almost, collectively. With this similarity in popular items, it may mean that the caches fail to retain the single most popular items in the cache but instead keep changing the items in the cache in an effort to retain popular items which could lead to a low cache hit ratio and therefore a higher average path stretch. This may never transpire as the cache may never be small enough in quantity for the top select few items of Zipf to exceed the top few items of Zipf-Mandelbrot.

3.4.3.3 KL Zipf & Zipf-Mandelbrot models vs. Empirical data-sets

The empirical data-sets for VoD and TV catch-up were admitted to the Icarus simulation environment as well as the theorised models thought to imitate the empirical data according to the KL divergence results listed in Table 3.3. The quantity of unique items in the Icarus simulation was set to be equal to the quantity of unique items found in the empirical distributions (≈ 11000 & ≈ 2500).

The topology chosen in the simulations was GEANT [66]. The Routing strategies adopted were "Cache less for more" [48], "Hybrid Symmetrical Hash" [51], "Leave Copy Down" [49] and "Probabilistic Caching" [51] which are representative of the best cache routing strategies implemented in Icarus. The average of all routing strategies makes up

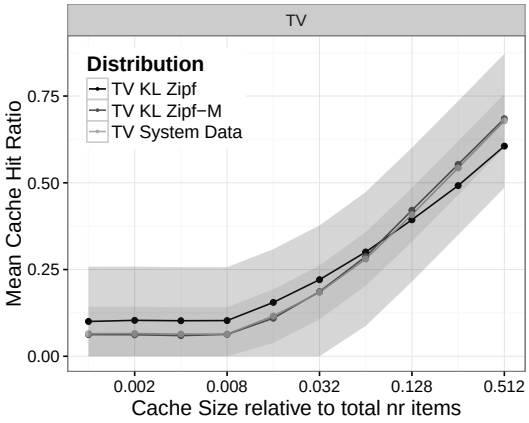


Figure 3.17: Icarus: Average Cache Hit Ratio for the BT TV dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

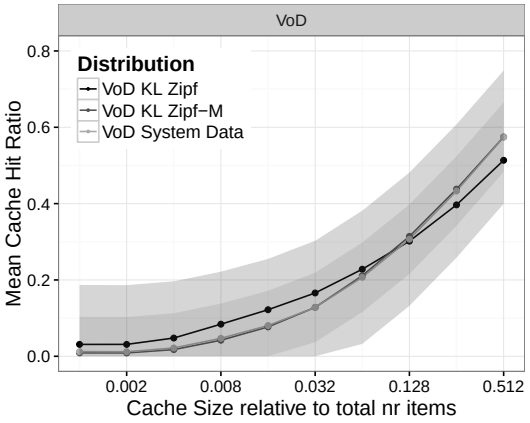


Figure 3.18: Icarus: Average Cache Hit Ratio for the BT VoD dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

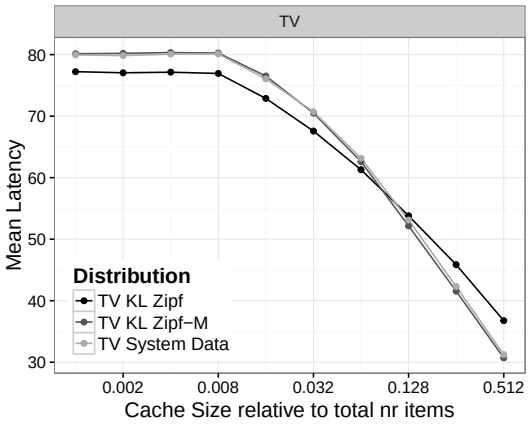


Figure 3.19: Icarus: Average Latency for the BT TV dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

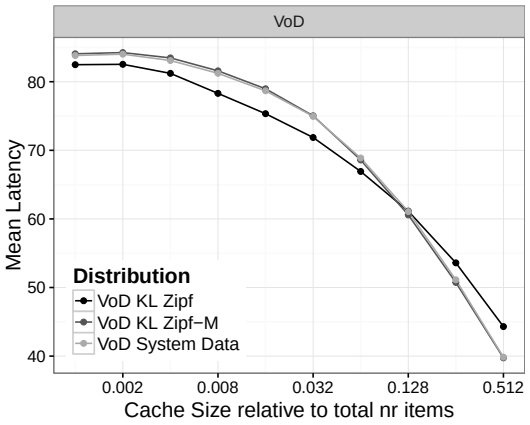


Figure 3.20: Icarus: Average Latency for the BT VoD dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

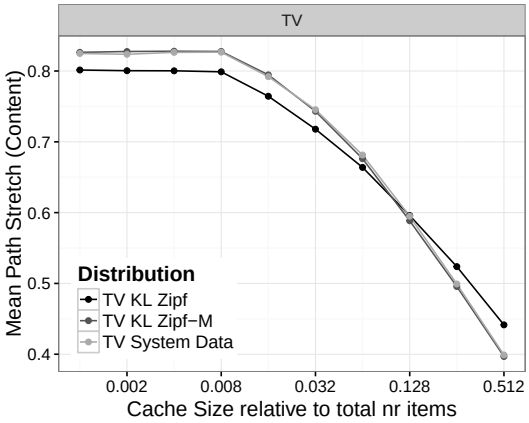


Figure 3.21: Icarus: Average Path Stretch results for the BT TV dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

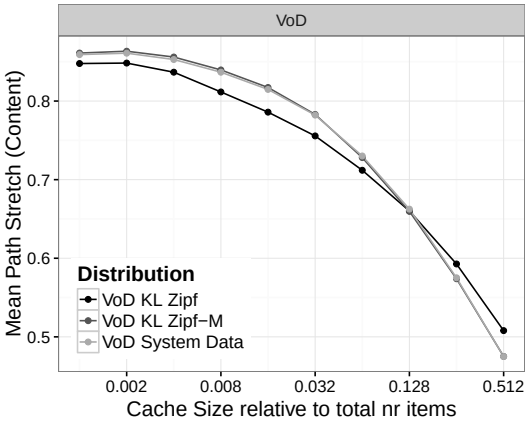


Figure 3.22: Icarus: Average Path Stretch results for the BT VoD dataset and the KL determined, closely matched, Zipf and Zipf-Mandelbrot distributions

the results displayed and presented. Ten cache sizes were chosen to test the probability distribution's differences for which the range is displayed in Equation 3.9.

$$\frac{2^{[0,1,2...,9]}}{1000}[11000, 2500] \tag{3.9}$$

Three measurements were obtained from each test with insights into the performance that can be expected from each of the 2 sets of 3 probability distributions tested. The measurements are *Average Latency*, *Average Path Stretch* and *Average Cache Hit-Ratio*.

The *Average Cache Hit-Ratio* results regarding the Empirical TV and VoD Datasets, as well as the KL deduced, best matching, Zipf and Zipf-Mandelbrot distributions can be found in Figures 3.17 and 3.18. The figures show, in the grey areas, the cache-miss hit ratios estimated per the method suggested in Chapter 3.2.4. Primarily it shows the estimated cache-hit ratio boundaries that can be expected for the Zipf and Zipf-Mandelbrot distributions in comparison to the cache-hit ratio that can be expected from a clairvoyant cache given the empirical probability distribution. The cache-hit error that could be expected from each Zipf and Zipf-Mandelbrot was calculated as stipulated in Equation 3.4.3.1. The darker, inner boundary estimating the cache-hit ratio relative to the empirical data cache-hit ratio if Figures 3.17 and 3.18 belong to the distribution found to produce the lowest KL value relative to the empirical dataset which was the Zipf-Mandelbrot distribution. The lighter, outer boundary estimating the cache-hit ratio relative to the empirical data cache-hit ratio belongs to the distribution found to produce the higher KL value relative to the empirical dataset which was the Zipf distribution. From the boundaries that give the estimated cache-hit ratio, as shown in the Figures illustrating cache-hit ratio, it is possible to surmise that, though the boundaries are broad, the cache-hit ratio boundaries for the given results are accurately estimated to be within the boundaries set.

The difference in cache-hit ratio for each individual model and the empirical dataset is shown in Table 3.6. The *Total Average* is not a measurement to take a final conclusion from as the cache-hit ratios measured, in relation to the empirical cache-hit ratio results, overlap on occasion and there are only ten points of measurement, meaning

	Empirical and Model — Mean Absolute Deviation			
	VoD		TV Catch-up	
Cache Size	Zipf	Zipf-Mandelbrot	Zipf	Zipf-Mandelbrot
0.001	0.0189	0.0034	0.0352	0.0027
0.002	0.0194	0.0031	0.0380	0.0034
0.004	0.0260	0.0043	0.0384	0.0044
0.008	0.0370	0.0050	0.0391	0.0008
0.016	0.0415	0.0036	0.0394	0.0062
0.032	0.0379	0.0002	0.0366	0.0021
0.064	0.0052	0.0074	0.0147	0.0128
0.128	0.0211	0.0043	0.0200	0.0070
0.256	0.0605	0.0007	0.0509	0.0101
0.512	0.0363	0.0050	0.0729	0.0061
Total Mean	0.0304	0.0037	0.0385	0.0056

Table 3.6: Difference between Cache-Hit Ratios relative to Empirical Results in test where Zipf and Zipf-Mandelbrot models were selected based on a "goodness-of-fit" produced in KL

a single low value may be over represented in the *Total average value*. The trend visible in Figures 3.17 and 3.18 where Zipf-Mandelbrot appears to follow a more closely related cache hit ratio over Zipf, together with a lower than Zipf *Total Average* cache hit ratio for Zipf-Mandelbrot in both scenarios, may leave one to conclude Zipf-Mandelbrot to be more closely associated with the empirical data for both TV catch-up and the VoD empirical sets than Zipf.

The Figures 3.19, 3.20 and 3.21, 3.22 show Latency and Path Stretch, respectively. They both demonstrate trends in network performance as subjected to the model representatives chosen, using KL fitting, which are Zipf and Zipf-Mandelbrot with a variety of cache sizes. The key performance indicator is the proximity to the empirical data *latency* and *path stretch* performance lines for each model, which for Zipf-Mandelbrot is hard to distinguish as it appears plotted extremely close to the empirical dataset as to be on top of the empirical dataset *latency* and *path stretch* performance lines. For this reason one may assume, as was assumed for the results found in terms of *cache-hit ratio* in the previous paragraph, the Zipf-Mandelbrot distribution is a more appropriate replacement for the Empirical data-sets over Zipf.

3.4.3.4 PCS Zipf & Zipf-Mandelbrot models vs. Empirical data-sets

The empirical data-sets for VoD and TV catch-up were introduced into the Icarus simulation environment as well as the theorised models thought to imitate the empirical data according to the PCS results listed in Table 3.2. The quantity of unique items in the Icarus simulation was set to be equal to the quantity of unique items found in the empirical distributions (≈ 11000 & ≈ 2500).

The topology chosen in the simulations was GEANT [66]. The Routing strategies adopted were "Cache less for more" [48], "Hybrid Symmetrical Hash" [51], "Leave Copy Down" [49] and "Probabilistic Caching" [51] which are representative of the best cache routing strategies implemented in Icarus. The average of all routing strategies makes up the results displayed and presented. The ten cache sizes chosen to test the probability distributions differences for which the range of which is displayed in Equation 3.9.

Three measurements were obtained from each test with insights into the performed that can be expected from each of the 2 sets of 3 probability distributions tested. The measurements are *Average Path Stretch*, *Average Cache Hit-Ratio* and *Average Latency* which aims to capture the delay experienced when content is traversing the network.

The *Average Cache Hit Ratio* results regarding the Empirical TV and VoD Datasets, as well as the PCS deduced, best matching, Zipf and Zipf-Mandelbrot distributions can be found in Figures 3.23 and 3.24. The figures show, in the grey areas, the cache-miss hit ratios estimated as per the method suggested in Chapter 3.2.4. Primarily it shows the estimated cache-hit ratio boundaries that can be expected for the Zipf and Zipf-Mandelbrot distributions in comparison to the cache-hit ratio that can be expected from a clairvoyant cache given the empirical probability distribution. The KL values are those given by subjecting the PCS found best representatives from each model in relation to the empirical datasets to the KL divergence test as stipulated in Equation 3.4.3.1. The darker, inner boundary estimating the cache-hit ratio relative to the empirical data cache-hit ratio if Figures 3.23 and 3.24 belongs to the distribution found to produce the lowest KL value relative to the empirical dataset which was the Zipf-Mandelbrot distribution. The lighter, outer boundary estimating the cache-hit ratio relative to the

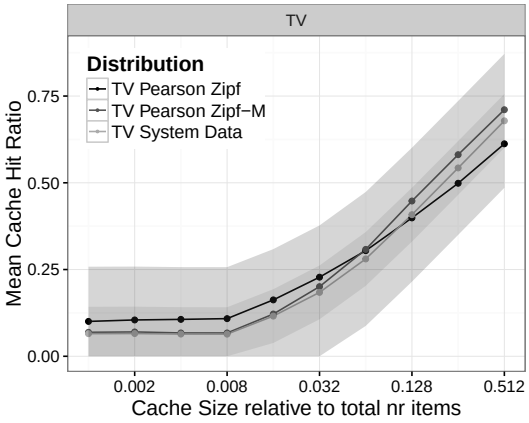


Figure 3.23: Icarus: Average Cache Hit Ratio for the BT TV dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

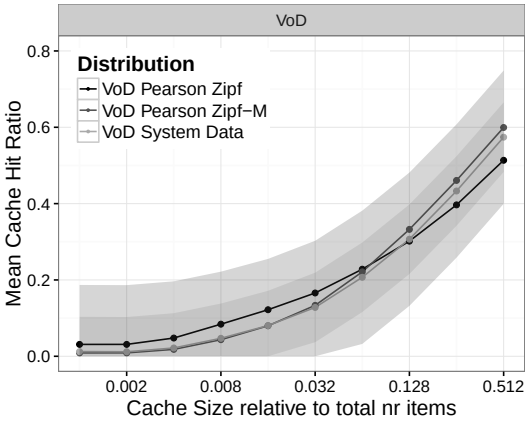


Figure 3.24: Icarus: Average Cache Hit Ratio for the BT VoD dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

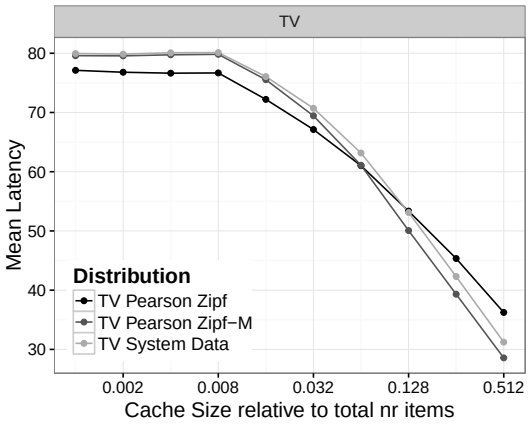


Figure 3.25: Icarus: Average Latency for the BT TV dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

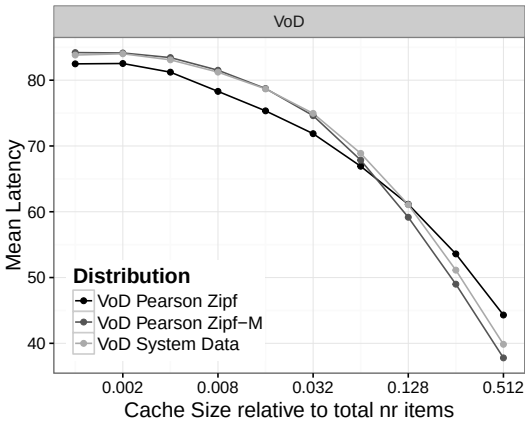


Figure 3.26: Icarus: Average Latency for the BT VoD dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

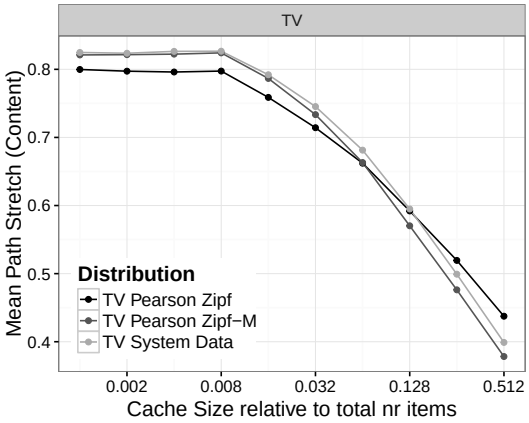


Figure 3.27: Icarus: Average Path Stretch results for the BT TV dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

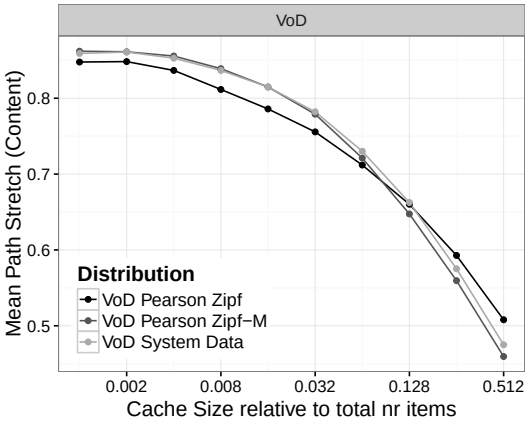


Figure 3.28: Icarus: Average Path Stretch results for the BT VoD dataset and the PCS determined, closely matched, Zipf and Zipf-Mandelbrot distributions

	Empirical and Model — Mean Absolute Deviation			
	VoD		TV Catch-up	
Cache Size	Zipf	Zipf-Mandelbrot	Zipf	Zipf-Mandelbrot
0.001	0.0189	0.0035	0.0353	0.0039
0.002	0.0194	0.0029	0.0392	0.0046
0.004	0.0260	0.0037	0.0423	0.0032
0.008	0.0370	0.0036	0.0447	0.0034
0.016	0.0415	0.0006	0.0467	0.0059
0.032	0.0379	0.0054	0.0438	0.0166
0.064	0.0052	0.0257	0.0092	0.0394
0.128	0.0211	0.0142	0.0238	0.0274
0.256	0.0605	0.0254	0.0442	0.0382
0.512	0.0363	0.0277	0.0663	0.0320
Total Mean	0.0304	0.0113	0.0395	0.0175

Table 3.7: Difference between Cache-Hit Ratios relative to Empirical Results in test where Zipf and Zipf-Mandelbrot models were selected based on a "goodness-of-fit" produced in PCS

empirical data cache-hit ratio belongs to the distribution found to produce the higher KL value relative to the empirical dataset which was the Zipf distribution. From the boundaries that give the estimated cache-hit ratio, as shown in the Figures illustrating cache-hit ratio, it is possible to surmise that, though the boundaries are broad, the cache-hit ratio boundaries for the given results are accurately estimated to be within the limits' set.

The difference in cache-hit ratio for each individual model and the empirical dataset is shown in Table 3.7. The *Total Average* is not a measurement to draw a final conclusion from as the cache-hit ratios measured, in relation to the empirical cache-hit ratio results, overlap on occasion and there are only ten points of measurement, meaning a single low value may be over represented in the *Total average value*. The trend visible in Figures 3.17 and 3.18 where Zipf-Mandelbrot appears to follow a more closely related cache hit ratio over Zipf, together with a lower than Zipf *Total Average* cache hit ratio for Zipf-Mandelbrot in both scenarios, may leave one to conclude Zipf-Mandelbrot to be more closely associated with the empirical data for both TV catch-up and the VoD empirical sets than Zipf.

The Figures 3.25, 3.26 and 3.27, 3.28 show Latency and Path Stretch, respectively.

They both demonstrate trends in network performance as subjected to the model representatives chosen, using PCS fitting, which are Zipf and Zipf-Mandelbrot with a variety of cache sizes. The key performance indicator is the proximity to the empirical data *latency* and *path stretch* performance lines for each model. It would appear that, regarding the lower cache sizes, Zipf-Mandelbrot closely resembles the *Cache Hit Ratio* observed by the empirical data-sets. As the cache sizes grow, the Zipf and Zipf-Mandelbrot Distribution appear to stray from the observed *Average Latency* and *Average Path Stretch* of the empirical datasets, however, the proximity of Zipf-Mandelbrot throughout the larger cache sizes is still more closely associated to Zipf. For this reason one may assume, as was assumed for the results found in terms of *cache-hit ratio* in the previous paragraph, the Zipf-Mandelbrot distribution is a more appropriate replacement for the Empirical data-sets over Zipf.

3.4.3.5 Conclusion Icarus KL vs. PCS

In Sections 3.4.2.1 and 3.4.2.2 permutations of the models Zipf and Zipf-Mandelbrot were deduced to be the distribution Φ that shared the greatest "goodness-of-fit" in relation to the empirical video popularity distributionΘ. The aforementioned Zipf and Zipf-Mandelbrot distributions Φ were subjected to the *Icarus* ICN cache-enabled network to determine if their behaviour would vary from the behaviour one may experience when the empirical VoD and TV catch-up distributions are subjected to the Icarus simulation environment. The key characteristics measured were *average latency*, *average path stretch* and *average cache-hit ratio*. The figures for the *Cache-Hit Ratios* for both the PCS and KL chosen Zipf and Zipf-Mandelbrot distributions Φ are assessed in relation to the empirical request distributions in Tables 3.6 and 3.7 and in Figures 3.23, 3.17 and 3.24, 3.18. The results of Tables 3.6 and 3.7 are repeated in Table 3.8 which shows a much closer *Total Average* in the for the Zipf and Zipf-Mandelbrot distributions produced by the KL fitting method over the PCS fitting method. Finally the *Mean Absolute Deviation* produced by the Zipf-Mandelbrot tests appears to be lower than the Zipf *Mean Absolute Deviation* results for both the KL and PCS deduced representa-

Mean Absolute Deviation Empirical and Models	VoD		TV Catch-up	
	Zipf	Zipf-Mandelbrot	Zipf	Zipf-Mandelbrot
Mean Absolute Deviation KL	0.0304	0.0037	0.0385	0.0056
Mean Absolute Deviation PCS	0.0304	0.0113	0.0395	0.0175

Table 3.8: Total Average Difference between Cache-Hit Ratios relative to Empirical Results in test where Zipf and Zipf-Mandelbrot models were selected based on a 11goodness-of-fit" produced in PCS and KL matching processes

tives for each model, concluding Zipf-Mandelbrot to be a more suitable substitute for the empirical dataset Θ.

3.4.4 Conclusion

In this Chapter we set out to identify whether the Zipf model or Zipf-Mandelbrot model shares a greater likeness to VoD observed request distributions. Two empirical VoD request distributions were supplied by BT to aid in identifying the most appropriate model. The methods used for measuring were *Pearson Chi-Squared* and *KullBack-Leibler* which each provided a best representative distribution of each model tested. Each representative was compared using the "goodness-of-fit" measurement produced by each tests. Each method of testing used deduced that Zipf-Mandelbrot is indeed a better representative than Zipf in relations to the VoD and TV catch-up video request distributions.

Additionally to the KL and PCS methods of testing and comparing, an ICN simulation environment with cache enabled nodes was used to measure the network behaviour of the empirical data in relation to the KL and PCS deduced best representative distributions from each model. The outcome from these tests highlights the measured characteristics (*average latency*, *average path stretch* and *average cache-hit ratio*) from the Zipf-Mandelbrot distributions share a greater likeness to the empirical VoD request distributions' measured characteristics when compared to the Zipf distributions. Concluding, Zipf-Mandelbrot from more closely modelled, the empirical data in terms of expected network behaviour in an ICN cache enabled network.

4

VoD Request Simulation Environment

A pseudo-realistic video request generator would provide valuable insight into video delivery systems. The target of identifying the spectrum of parameters required to simulate such an environment applying pseudo-realistic parameters that would accurately estimate the request behaviour for a specific demographic is challenging. The parameters can be identified in video delivery systems currently already deployed or through surveys of a specific area / groups identified as the target audience. The parameters would include such things as: the quantity of videos at any time; lifetime of videos on your system; decay of video popularity of your system; distribution of video popularity; daily / weekly distribution of request relative to time of day and the number of total active users of your system. These are just some of the variables that go into generating realistic user requests. This Chapter sets out to design and implement such a VoD request generator accounting for all the above parameters.

4.1 Introduction & Motivation

The ability to simulate video requests with reasonable accuracy would provide an asset to the Video on Demand market that has become the majority of internet traffic in present days [1,6]. A video request simulator would equip video content providers with a tool to predict, and thus resolve, networking and application platform issues before they arise when the cause of the issues stem from the design of the system in question. In Chapter 5 a cache eviction algorithm is introduced for which a request generator, such as the one in this Chapter, would provide valuable insight into the behaviour of such an algorithm.

Users of Video on Demand systems appear not to exemplify a drastic range of behaviours, though still complicated as many environmental variables are at play at any one moment. The increase of popularity per item when introduced, the decrease of popularity of items once their maximum has been reached, the cumulative request total per item, the request quantities throughout any given time are all factors that can be broken down to be included in a single simulator which is what is attempted in this chapter.

One limitation of the pursuit to produce a simulator that encompasses all the variables mentioned is the lack of quantifiable research data that can be used to produce such a simulator. A number of research papers have attempted to find and quantify the parameters required, however, they appear to forget to consider the problem in its entirety, which limits the possibility of simulating VoD data accurately. An example of this is the decay of video objects which has been researched [16,36,37], however, without also considering growth of popularity video objects, this research is incomplete.

Popularity distributions of VoD systems are also important to consider and are found in many publications [9,10,14,17,18,22,33,38,70,71]. They provide a brief glance (typically ranging from a week to a number of weeks) into observations of the popularity of all items in a VoD system (often found to be approximately Zipf, however debated to be Zipf-Mandelbrot in Chapter 3).The primary limitations of these observations are

that new items are introduced in this period and items entered prior to the start of the observation are only observed in their decayed state. The popularity distribution not measured is the total request distribution for all items at a single moment in time from their conception to removal ("birth" and "death"). Such a probability distribution would provide one with the request distribution of all items at a single moment in time. The items observed may be in the system for various lengths of time, however, it would give a more complete picture of the popularity distribution one can expect.

4.2 Breakdown of parameters

The parameters discussed in this chapter are those that will aid in the creation of the simulator of Video on Demand traffic for an entire system. The number of parameters considered are not all the parameters found in Video on Demand systems. This project would be of an unfathomable scope as network limitation, application front-end restrictions [17], data storage, client limitations, and many more would all start becoming factors in simulating.

4.2.1 Parameters included

4.2.1.1 Request Distribution

The request distribution of VoD systems represent the popularity of items present in a system, usually over a set period of time. Popularity distributions for a variety of systems has been released and published among many journals and research papers in the context of VoD [7–12, 12–18, 22, 33] (discussed further in Chapter 2.1.2.1). The request distribution referenced would not suffice for the simulation environment developed as the method of observation is limited to a period of time during which items were introduced and removed. The popularity measurement required for the simulator would require knowledge of all items' total request quantities, from introduction of said items, until the items' expiration. A popularity such as this would provide a more accurately description of the items without the effects of decay creating a disruption to

the observed data.

An assumption can be made that the total requests per video PDF acquired in a set window of sufficient size is approximately equal to a total requests per video distribution. If this is assumed then the probability distributions in Chapter 2.1.2.1 would be sufficient for the initial introductory probability distribution of items in the system.

A distribution generator in the form of a PDF generator accounting for variable length is included to ensure flexibility for all video systems. This is done in the form of a Zipf-Mandelbrot PDF generator (which with a ν variable of 0) also can generate Zipf-like distributions. The Zipf-Mandelbrot parameters used are those listed in Chapter 3. Alternatively a pure Zipf Distribution may be used for the purpose of demonstration of the simulator developed.

4.2.1.2 Decay Rate

The objective of the simulator is to, on a large scale, simulate all video items throughout a finite amount of time. This requires that items experience a changing popularity on the basis of time.

The subject of decay is discussed by Gummadi et al. [16] in the context of a peer-to-peer system called "Kazaa" in a rather poor manner as items larger than 100 MB are declared to experience a larger amount of request for items older than one month by 72% and items smaller than 10 MB are declared to experience a larger amount of requests towards items older than one month but by only 52% with the omission of medium sized items. The decay here is for multimedia items, not exclusively VoD, which causes some doubt as to how much this data resembles exclusively VoD data. It may well be possible that a majority of items observed are not video, dictating the decay-rate experienced by items in the system.

The subject of decay is also discussed by Li et al. [36] and Chen at al. [38] appear

to agree that within their observed datasets that the initial day of observations see the videos decrease in popularity by 20%, and again decrease by 20% 9 days later. This observation is based on video request data that is explicitly not UGC on both P2P based systems, as well as CDN run systems, such as Hulu and PPLive. Li et al. [36] appear to consider items once they are stored on the cloud device and then observed the decreasing popularity. This comes after a video is selected to be popular enough to be submitted to the cloud, which may see a large part of the item's lifetime be omitted (an hour minimum due to design), as the time before the item became popular it was not yet considered for measurement of popularity, especially considering the initial surge in popularity.

Chen et al. [39] follow the lifetime of video items in a system which includes the decay. The observation is made in a VoD service run by Tencent, which is one of the largest VoD services in China. It provides video to an active user base of over 50 million people. For the purpose of measuring video decay, videos are separated into categories, easily distinguishable by key characteristics (Movies, Music Videos, TV content, News and Sports), for which each category has a total decay rate measured over the duration of 7 days. The observations made suggest that most categories appear to see a great quantity of requests on the day of release, as well as the day following. After this period all categories see a drastic decrease in popularity, with the exception of the "Movies" category which appears to see a peak in interest on the third day the item is in the system. It is unclear if items on day 0 (the initial day of release) were in the system for the entirety of it, or just the later section of the day. This additional bit of missing information would point out if the items, almost immediately after release, start seeing a decreasing request rate or if this happens, as suggested by Chen et al., on the second day in the system. The conclusive statement by Chen et al. [39] is that video items in their infancy receive a great amount of promotion, mainly on the front-page of the VoD application, which may be the primary reason for the popularity received by these newly introduced video items.

VoD System	Nr. of Observed Entries	τ	α	β
YouTube (Japan & USA)	≤ 160	≤ 20	≈ 0	0.5
UitZending Gemist	≤ 42	≥ 10	$\approx 10^6$	0.5
IMDB	≤ 18	≥ 10	$\approx 10^6$	0.5

Table 4.1: Results gathered by Avramova et al. The results are approximated from the graphs in the publication which can be factored in to the simulator

Avramova et al. [37] provide a close look at the decay of video popularity of primarily video items which have already achieved a great deal of popularity by their submission to the top 50, list on their respective platforms, such as YouTube (US and JP platforms), IMDB rental records, and "Uitzending Gemist(.nl)" which is a Dutch TV catch-up service. Once submitted to the top 50 the items were observed for a period of a minimum of 30 days. The observations concluded that it is primarily the UGC that sees a thick tail in the observed decay curve which can be described as a power-law distribution. All other traces, described as TV catch-up services, appear to follow a more drastic decrease over time described as following exponential decay. The formula used by Avramova et al. [37] to plot the decay in the form of a CDF experienced by video items in all systems is as follows:

$$I_k(t) = \rho_k \left[1 - \left(1 + \frac{\left(\beta^{\frac{-1}{\alpha_k}} - 1 \right) (t - \Theta_k)}{\tau_k} \right)^{-\alpha_k} \right]$$

For which τ describes the time it takes for a fraction $1 - \beta$ of the total view count to accumulate. Δt describes the time at which the item is observed and Θ describes the time of entry in the system. α is the important variable to consider as it is the variables that is form-determining in the function. If α is large the function produces an exponential curve which would see the item gain the majority of its requests in the beginning of its existence. If α is small the function produces a power-law curve which would see the item remain relevant for an extended period of time, with a small amount of popularity remaining throughout the item's existence.

For the tests performed the variables most likely to simulate each category are as in Table 4.1 [37]. The variables chosen for the purpose of testing this simulator are

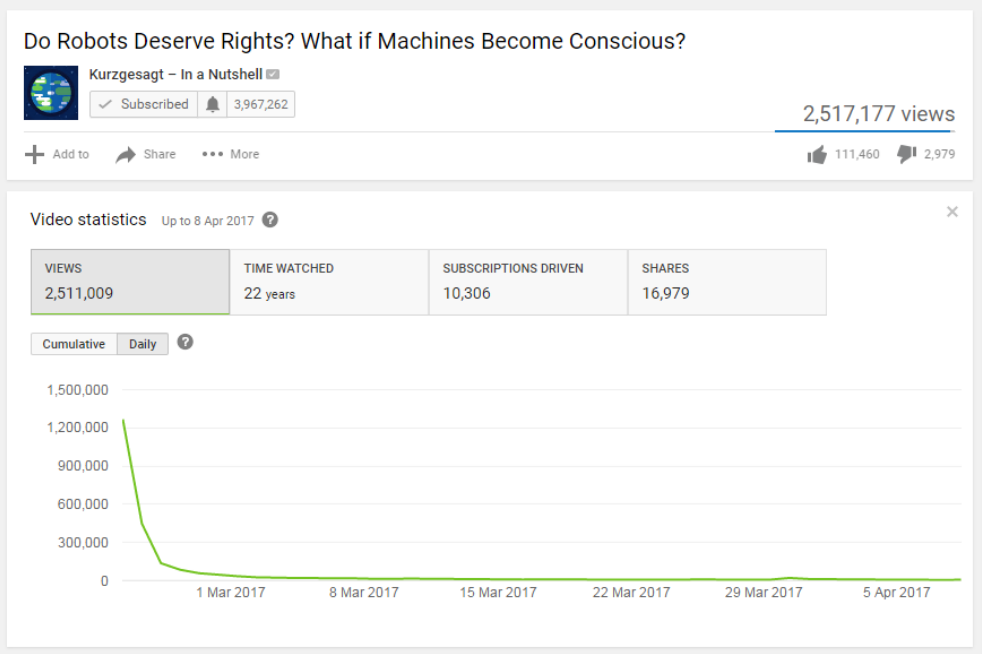


Figure 4.1: Single YouTube Video demonstrating decay over the duration of a month observed on 10/04/2017 with a total view count of 2,511,009. The observation and screen capture was made by Hildebrand Weisenborn, the author of this Thesis

$\alpha : 1, \tau : 40 and \beta : 0.5$.

The video popularity evolution for a UGC system such as content on YouTube [72] can be accessed publicly. This provides us with insight into the daily evolving popularity of videos published on the system. The YouTube platform can be linked to and referenced from foreign websites as well as local users to YouTube who can subscribe to channels they personally have a preference for. The videos from channels subscribed to by the user are presented to the user when the user accesses the YouTube website. Informing users of videos they will likely be interested in viewing soon after the video is made public is likely the cause for most views being at the start of the videos existence as demonstrated by a single sample case is Figure 4.1 which are the statistics for a video that has been publicly available for over a month. The video in Figure 4.1 is posted on the YouTube Channel ("Kurzgesagt In a Nutshell") to which a large amount of users are subscribed (On 10/04/2017: 3,967,384 subscribers).

4.2.1.3 Lifetime

The lifetime of a video signifies the time video objects spend in a system available for request. The lifetime of videos in a system is dictated by the design of the VoD system for which frequently UGC and non-UGC hold great disparity.

It is observed in UGC systems such as YouTube [72] that videos have a date of introduction and not a date of removal. Removal of video objects may still happen if a user decides to remove a video, however this action is not systematically exercised by individuals on the YouTube platform.

Non-UGC VoD systems may vastly range in the length of time videos are present in the system. In the case of the data in this Thesis that was provided by BT, the TV catch-up data had an average lifetime of two weeks, which was drastically different from the VoD data provided which saw videos remain in the system for a period far exceeding the TV catch-up data by potentially months. In the cases of a number of British public broadcast channels which make their content available online or provide all content of other providers online; BBC Iplayer [73], ITV [74], Channel 4 [75] and Channel 5 [76], they provide videos that are made available for streaming for the duration of 30 days or fewer. Sky Go Extra [77] provides a VoD service with content that is available for 30 days, however they also provide a service for Movies which are available for an undefined amount of time. Channel 5 [76] provides content commissioned by Channel 5 to be available for the period of a year or more, thus highlighting the variability of VoD systems.

For the purpose of the simulations, we assume most VoD services which provide TV catch-up services (with specific interest in the UK) appear to make content available for the period of 30 days. For content that constitutes primarily of movies, it appears there is no defined period of time after which content is removed.

4.2.2 Parameters not considered

4.2.2.0.1 Item Popularity Growth is not included as the initial growth of items is a parameter for which currently a very limited amount of data is available in the

research community in the context of non-UGC systems. The assumption that new video items have the greatest quantity of requests on release, and soon after, is thus applied in this simulator. This restriction may not stray far from a simulation where growth is considered if the increments of time in the simulator are of large enough size [39] (e.g. the peak of the item popularity may be achieved in the first 6 hours of existence, thus if a single sample period is equal or greater than 6 hours, the simulation may be approximately equivalent as with the item growth included).

4.2.2.0.2 Individual User Requests is not included. Quantifying the number of users, accompanied by the quantity of requests per user, is not considered for this simulation. It simply considers the total sum of all requests in a single system without the consideration for where they may be destined. The justification for this is the granularity required to objectively quantify the localisation for each group of users and that the localisation bias introduced for each localised group would be hard to implement and would complicate the simulator significantly. Instead the assumption is made that, as long as each group is large enough in size, each user group is homogeneous, expelling the need for localisation.

4.3 Simulator Breakdown

The simulator is designed to imitate the requests received by a VoD system as given by the literature review above. The specific individual user behaviour and locations, as well as the network design of the CDN delivering video data, is not considered. Focussing purely on requests received by a large group of users provides with the means to apply a topology and simulated environment of video data later if this is available. Scalability can be tested with consideration of evolution of the video delivery system over time, as well as an increasing user basis and therefore an increasing request quantity. The structure used to simulate the popularity rise and fall of individual video objects, as well as the introduction and removal, in the context of a specific system can be broken down into two processes; Storyboard Generation and Request Generation, described below.

The initial process, "Storyboard Generation," generates the script that is provided by the data the user describing each video object. The script provides us with a chronological set of video objects at each Δt and their respective request probability at the aforementioned Δt throughout T. The script is comprised of the probabilities of request to all items in the system at Δt in T as well as the time the objects remain requested in the system described by all Δt the object remains mentioned in the script.

The second process, "*Request Generation*," takes as parameters the script created by the initial process "Storyboard Generator" and converts it into a linear set of requests in the order of requests to the VoD system. the script generated by the "Storyboard Generator" as Δt specified as the step base time identifier as to when decay happens. Each Δt may be specified as to mean a single day or a single hour, whichever the user requires. The quantity of requests for each Δt may be chosen by the user to reflect the simulated system specifics. The quantity of requests to each Δt may vary, as would be expected if Δt is set to simulate an hourly basis.

More details for each process will be described below in the chapters titled "Storyboard Generation" and "Request Generation".

4.3.1 Storyboard Generation

The Storyboard describes the script that is generated from the data the user provided. The script provides us with a chronologically ordered list of video objects and their respective request probability.

The parameters required to generate the script are in the format demonstrated in Table 4.2. Table 4.2 is comprised of the total number of video objects in the VoD system at any single moment in time. Each video object has a number of parameters associated with it which are decay rate in the form of a function, the total time each video object will be in the system for and the total request probability of each video object. For

Item ID	Life-Time	Decay Rate	Total Probability
1	24	Function X	0.005
2	168	Function Y	0.002
3	72	Function Z	0.04

Table 4.2: User supplied data for generating sudo-realistic requests

the purpose of condensing the number of items, it is possible to consider that once a video object is removed, no more requests are made to that object, thus shortening the Life-Time attributed to that video object.

Once the initial data has been entered, as stipulated in Table 4.2, it is processed. The first part of the script generated will function as a "warm-up period". The purpose of the "warm-up period" is to gradually introduce items as to ensure not all items in the system are made available at the same interval. The "warm-up period" is a part of the script that can be viewed as dismissible in the tests as during this period the system will be dominated by the start up effects. The length of Δt required for the "warm-up period" will be equal to the "Life-Time" of the video object with the greatest "Life-Time".

The script holds the probability distribution of each video object at each Δt in the system. The size of ΔT (total time) in the system is not influential in the quantity of video objects in the system at each individual Δt as each time an object is removed, a new video object with the same characteristics is reintroduced. The decay of probability of request is set to be the original request probability of the removed item. The new item could be considered a clone of the removed item with a new ID as to mark it as a new item with the same characteristics as the removed item.

Table 4.3 is a short example script which includes the fields as produced by the Storyboard Generation program. The example shows two items introduced with the third still waiting for introduction. This period would be considered the "warm-up period" and is comprised of the exact same structure as the period after the "warm-up period". One is required to remember the value Δt which signified the length of the "warm-up period" as only when Δt is greater than "System Time" is the "warm-up period" over and can observations begin in terms of the tests.

System Time	ID	time in System	Total time	Decay Rate	Total Probability	Current Probability
1	1	1	24	Function X	0.005	0.003
1	2	1	168	Function Y	0.002	0.015
2	1	2	24	Function X	0.005	0.001
2	2	2	168	Function Y	0.002	0.003

Table 4.3: Storyboard for generating sudo-realistic requests

4.3.1.1 Reintroduction of items

Once an item expires it is removed. Once an item is removed it is considered to have decayed to the extent that it will no longer generate any request. The assumption that items may not receive any requests in the future may not reflect what we may expect in a real scenario, however keeping items with infinitely small probabilities of requests would be become an increasing constraint when generating request probabilities for many Δt. Additionally, video providers may limit the time during which an item is available. Once an item is removed - a clone is created of the removed object that holds the same properties (Decay Rate, Total Probability, Life-Time) but with a different ID. The clone of the removed item is introduced back into the system as a new item immediately after the removal of the original item with the same properties; creating infinite removal and reintroduction of all video objects as their life-time is reached in the system. In Figure 4.2 the total probability of items is shown when they are introduced into the system. A constant "Life-Time" value is chosen across all items in the VoD system to demonstrate a reoccurring item pattern. Figure 4.2 demonstrates the "warm-up period" during which life-time in the figure is uniform. Items may expire at Δt increases before all other items are introduced, creating an irregular introduction pattern during the "warm-up period". The length described as "Number Δt before reintroduction" signifies the Δt quantity of "Life-Time" of items in the system. The items highlighted by the coloured line signify all items which will be in different stages of decay at a single Δt. It should be noted that a different seed is used each time the simulation is run, thus creating a unique warm-up period in each iteration. The repetition seen in Figure 4.2 is only visible due to the uniform "Life-Time" specified across all items in the system as non-uniform

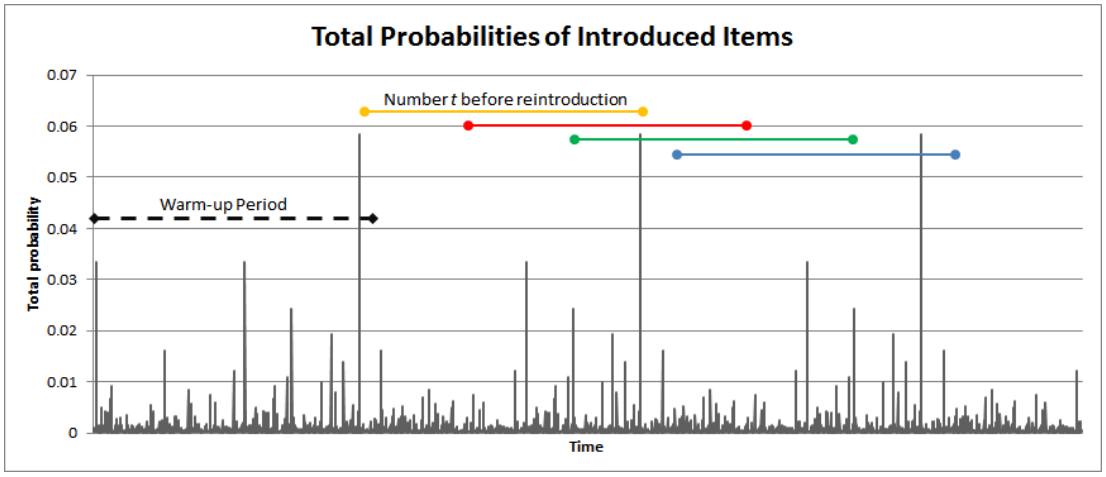


Figure 4.2: Total Probability of items when they are introduced into the system over time

"Life-Time" would see some objects go through a number of removal and reintroduction cycles while other go through fewer removal and reintroduction cycles.

4.3.1.2 Request Probabilities at Single Δt

Figure 4.3 illustrates the request probability of each item in the system at any Δt (after the warm-up period). The index of each item is listed on the horizontal axis for which the *Total Request Probability* and the *Current Request Probability*, which has the decay function developed by Avramova et al. [37] applied, is plotted. The horizontal axes have been ordered to reflect the *Current Request Probability*. The Avramova et al. [37] developed function has been applied to reflect items in various stages of their total life-span meaning the *Current Request Probability* may reflect items that have recently been introduced (high *Current Request Probability*) and those who have been in the system for a longer period of time (low *Current Request Probability*).

Figure 4.4 illustrates the request probability of each item in the system at any Δt (after the warm-up period). The index of each item is listed on the horizontal axes for which the *Total Request Probability* and the *Current Request Probability*, which has the decay function developed by Avramova et al. [37] applied, is plotted. The horizontal axes have been ordered to reflect the *Current Request Probability* and the *Total Request Probability*, irrespective of each other, meaning that the indexes for Current and Total

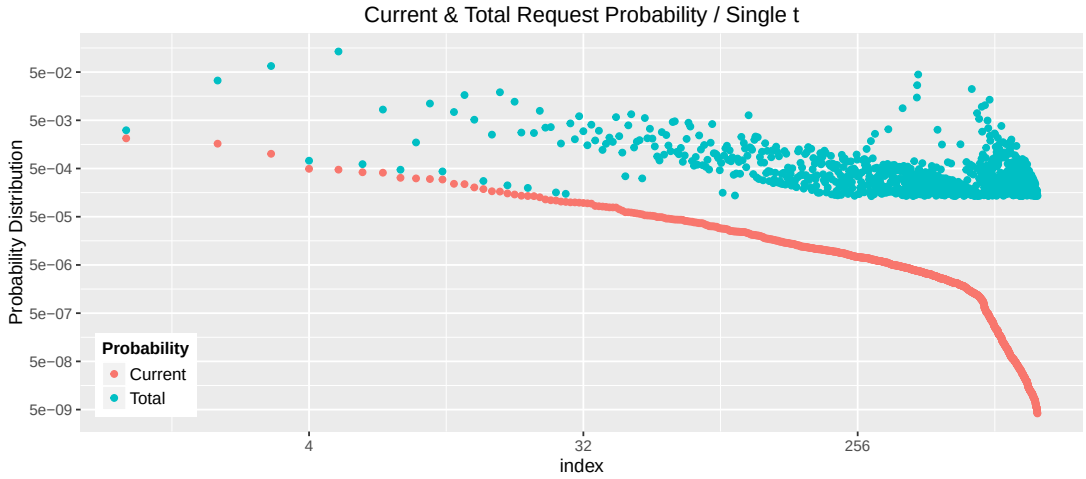


Figure 4.3: Probability of items at single Δt (Current & Total) — The Index refers to the same items in terms on Current and Total probability.

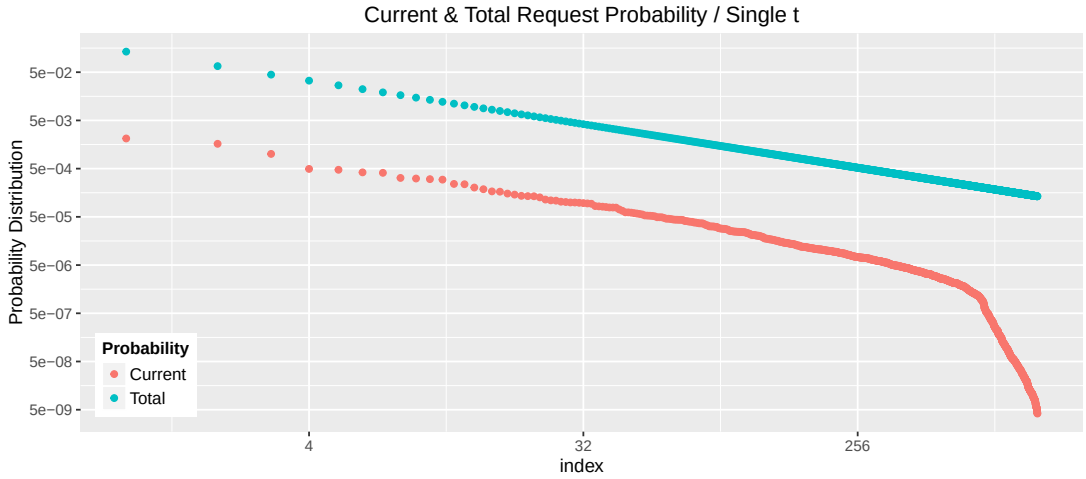


Figure 4.4: Probability of items at single Δt (Current & Total) — The index for the Current & Total probabilities are not referring to the same items.

Request Probabilities do not reflect the same items. In this figure, the separation between the *Current Request Probability* and the *Total Request Probability* can be seen, which brings to attention an interesting difference in the shape of each distribution and illustrates the effect decay may have on a system to which items are introduced on an incremental basis.

4.4 Request Generation

The *Request Generation* process takes as input the script generated by the *Storyboard Generation* process. The *Request generation* process then creates from the script an ordered list which holds the request IDs in the order they would be requested from the VoD system/CDN. Δt for each item can be included.

The *Request Generation* process iterates through the storyboard, selecting all entries for each "system time" and generating requests based popularity distributions for each Δt that is reflected by the "Current Probability" values. The request based popularity distribution for all Δt should be normalised to 1. Once normalised the probability distribution can be multiplied by the total request quantity of each Δt which can be dictated by the user as to simulate a day-by-day request distribution. All example requests shown in this chapter use a uniform request quantity for all Δt.

The probability distributions displayed in Figure 4.5 and 4.6 illustrate the request frequency of all items sorted by observed frequency for specific lengths of Δt not including the *warm-up period* requests. The example simulation displayed in Figures 4.5 and 4.6 have the following properties:

1. 1000 items in the System at any single Δt.
2. Items have a decay function with the parameters specified in Chapter 4.2.1.3 (An implementation of the function introduced by Avramova et al. [37]).
3. Items have a lifetime of 30 days.
4. A single Δt receives 10000 requests.
5. *Warm-up Period* is equal to $30\Delta t$.
6. *Main Request Period* is equal to $60\Delta t$.
7. The *Total Request Probability of all items is equal to Zipf($\alpha = 1$)*

The Request Probabilities in Figure 4.5 illustrate the request probability distribution observed when time observed is large, in this case $15\Delta t$. The *Total Request Probability*

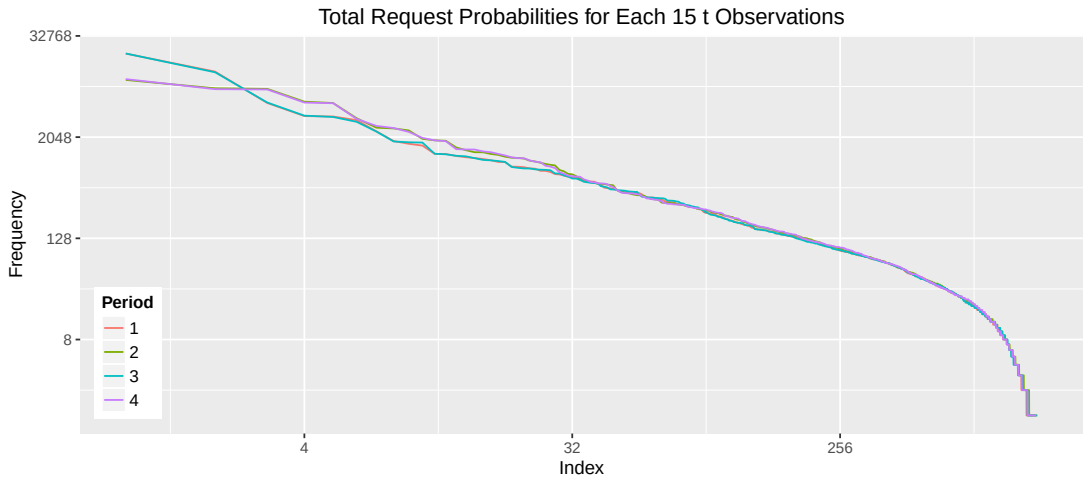


Figure 4.5: Probability of items at intervals of 15 Δt. Observations were made 4 times accounting for each Period as shown in the legend.

of all items is set to Zipf($\alpha = 1$), however the observed request distribution appears to follow a Zipf-Mandelbrot distribution as is suggested to by the video request probability in Chapter 3. For this reason a test was performed to identify if the distributions observed in Figure 4.5 are Zipf or Zipf-Mandelbrot. This was achieved by generating a large number of varying Zipf-Mandelbrot distributions which were then supplied, accompanied by the observed distributions in Figure 4.5 to a KL test. The test was performed for every period that is specified in the figure. The lowest KL results (Best fitting) are stored in Table 4.4 showing that indeed a Zipf-Mandelbrot distribution was found, however very closely resembling Zipf. This is so, due to the low value of ν. If ν is equal to 0 a Zipf distribution is concluded. Another set of tests were preformed on the popularity distributions produced when the period over which popularity distributions were observed is equal to $6\Delta t$. The results are shown in Table 4.5 and illustrated in Figure 4.6. All distributions are Zipf as $\nu = 0$ in all cases, thus the assumption can be made that, as time progresses and items are introduced and removed with decay, the popularity distribution of all items observed appear increasingly more Zipf-Mandelbrot like due to the similarity in items removed and introduced.

Period	KL value	Alpha	ν
1	0.0188	1.05	5
2	0.0286	1.25	5
3	0.0345	1	0
4	0.0287	1.25	5

Table 4.4: Observations for every 15 Δt matched to a Zipf / Zipf-Mandelbrot Distribution using Kullback-Leibler divergence as the method of matching

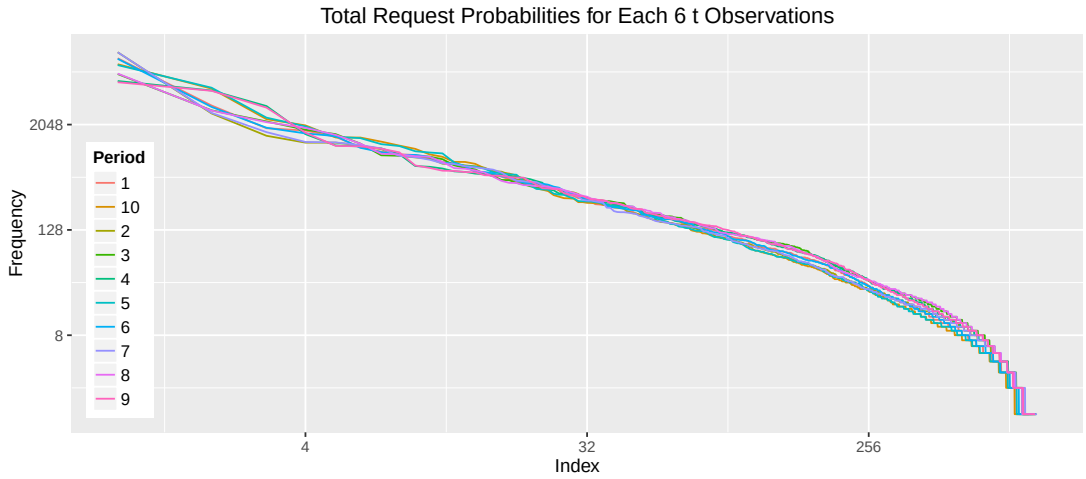


Figure 4.6: Probability of items at intervals of 6 Δt. Observations were made 10 times accounting for each Period as shown in the legend.

Period	KL value	Alpha	ν
1	0.0417	1.05	0
2	0.0559	1.1	0
3	0.0410	1	0
4	0.0464	1	0
5	0.0274	1.1	0
6	0.0432	1.05	0
7	0.0524	1.1	0
8	0.0402	1	0
9	0.0473	1	0
10	0.0270	1.1	0

Table 4.5: Observations for every 6 Δt matched to a Zipf / Zipf-Mandelbrot Distribution using Kullback-Leibler divergence as the method of matching

4.5 Conclusion

This Thesis chapter set out to create an ordered list of request for video objects made to a CDN that reflect the changing probabilities one may expect to find in a VoD system.

The key characteristics that were set out to include for each unique video object were; *Probability of Request*, *Decay of Popularity* and *Life-Time*. The process suggested to create a list of requests requires the user to introduce the properties of items in the system at a single point of observation. Once the properties of each item are identified, the requests can be generated by a two step process. The steps are documented as being the "StoryBoard Generator" (Chapter 4.3.1) and the "Request Generator" (Chapter 4.4).

The list of requests generated by the request generator here may be of use to a number of ranging application requiring a pseudo-realistic request order for items. One such example may be cache eviction policy effectiveness in a VoD setting such as for the algorithms proposed in Chapter 5. To conclude, a successful pseudo-realistic request generator for a VoD system was designed and implemented for the purposes specified.

5

Bubble Cache Eviction Algorithm

5.1 Abstract

In this chapter a novel algorithm is introduced. The algorithm is called the Bubble cache eviction algorithm, which is in reference to the Bubble sort algorithm, [21] which it resembles in some of its behaviours. The introduction of Bubble, together with the introductions of variations of Bubble, is intended to provide an alternative algorithm to the known cache eviction algorithm body of work such as LRU, FIFO, RAND and LFU.

5.2 Introduction

Cache eviction algorithms are used in combination with computer software or hardware storage to temporarily store for the benefits and availability of the environment in which they are implemented. The most commonly used cache eviction algorithms are simple, low operation cost algorithms such as LRU, FIFO and RAND with operational costs of $O(1)$ per "GET", "SET" or "EVICT" operation in a HashMap implementation. LFU is another cache eviction algorithm which is a frequency based algorithm with an operation cost of $O(\log(n))$ for "GET", "SET' and "EVICT" operations in a LinkedList implementation. In this Chapter, an alternative novel algorithm named "Bubble" is introduced with operational costs similar to LRU, FIFO and RAND. The functional strengths and weaknesses of all the algorithms mentioned will be explored in the following sections. A number of tests are constructed, utilising the Markov Chain analysis as well as a number of simulation methods are used, to enable comparison of the algorithms under a variety of circumstances that each provide insight into a realistic system such as a VoD system or other request system with similar request behaviour.

5.3 Known Algorithms

5.3.1 LRU

Least Recently Used [78–82] LRU is an eviction algorithm that works on the premise that the most recently requested items should be upheld as the most probable items to be requested again. This means that the item recently requested is pushed to the top of the cache, thus pushing all other items in the cache down one space. If the number of items in the cache exceeds the cache size the item at the bottom of the cache is evicted.

The largest operational cost for LRU is the method by which the search is performed to identify if an item exists in the cache. If a perfect HashMap is assumed (no conflicting entries) then it is safe to presume an operational cost of O(1) to perform the search for existing items. The operation of adding / removing is also O(1) thus assuming that

LRU has an overall operational cost of O(1).

e.g. LRU, we have an index of items $I = \{i_1 \dots i_M\}$ which are unordered, a subset of the items are in a cache index which is an ordered set which at time t is given by $C^t = \langle c_i^t \dots c_N^t \rangle$ where at any one point in time a subset of the items I is associated with the cache index e.g. at time t_1, it could be that $c_1^{t_1} = i_5, c_2^{t_1} = i_8, \dots$. Then at time t (where here, for simplicity's sake, time discrete integer points will be made) generally, the algorithm works such that if item i_k is requested and newly added to the cache then $c_1^t = i_k, c_2^t = c_1^{t-1}, c_3^t = c_2^{t-1} \dots c_j^t = c_{j-1}^{t-1} \dots c_N^t = c_{N-1}^{t-1}$; finally item c_N^{t-1} is evicted from the cache. Item i_k is fetched from the source (or another cache). In the case where the requested item i_k is in the cache at c_x^t then $c_1^t = i_k, c_2^t = c_1^{t-1}, c_3^t = c_2^{t-1} \dots c_j^t = c_{j-1}^{t-1} \dots c_x^t = c_{x-1}^{t-1}$ and all items $c_x \dots c_N$ remain then, unchanged.

5.3.1.1 Strengths

LRU benefits from the possibility of rapid eviction of items if those items no longer receive requests (cache hits) as the minimum number of requests required to reject an item from the top most position in the cache is equal to that of the cache size. LRU holds each item that receives a request to be the most important item at that time. This means that newly introduced items that will indeed generate a large number of requests in the future are likely to be in the cache and remain in the cache from the moment of their first request.

5.3.1.2 Weaknesses

LRU holds newly requested items to be the most important at that specific time. This entails that items which may experience a request only once and then see no further requests are upheld in the cache for an equal number of requests as is the size of the cache. For this duration, the space is arguably wasted and would greatly benefit from occupation from an item that would experience a larger amount of requests.

5.3.2 FIFO

First-in-First-out [78, 79] (FIFO) is an eviction algorithm that functions like LRU with the exception that an item that is in the cache and receives a request (cache hit) is not moved to the top of the cache but instead remains in its current position, not changing the state of the cache and the items currently held within it. Items can only be submitted to the top of the cache, considering them most important, when the item receiving the request is not yet submitted to the cache. This will remove the item at the bottom of the cache if the cache is not yet full.

The operational cost of FIFO may be largely affected by the search for items already in the cache which is the same as mentioned for LRU. This cost may be assumed to be O(1) in a Perfect HashMap implementation. The operation of inserting and deleting is exactly O(1) due to the ordered, fixed length nature of FIFO.

5.3.2.1 Strengths

The Strengths of FIFO are the same as LRU with a slightly slower rate of eviction due to the restricted movement of items held within the cache.

5.3.2.2 Weaknesses

FIFO has a more static pool of items with the elimination of movement of items within the cache. Frequently requested items in the cache will not be moved to the top of the cache when requested in succession and will be evicted once sufficient other items have received requests. This creates an environment where items, which in LRU would never be removed from the cache, would in FIFO see the occasional drop from the cache due to the new entries from outside the cache replacing the top most item and flushing the rest of the cache out.

5.3.3 LFU

Least Frequently Used [78–83] (LFU) is a frequency based algorithm that records the frequency of item requests and of items held in the cache. The order of those items

is according to the observed items request count, thus keeping the most frequently requested items at the higher end of the cache, thus assuming they are the most popular. The less frequently requested items are held in the bottom of the cache and thus, are the items that will be evicted first in case a new item receives requests. The implementation of LFU considered for the simulations has all items be reduced equally and ensures that the lowest possible item is always low enough in request count to be replaced by items which see new requests and are not yet submitted to the cache.

The operational costs of LFU is increased over LRU and FIFO as the a LFU cache requires ordering based on the quantity of requests received per item. The operation of "GET", "SET' and "EVICT" are of operational costs of O(log n) [84].

5.3.3.1 Strengths

LFU benefits from the memory of previous requests. In the case of popularity distributions such as Zipf, the popular items see a much greater amount of requests in contrast to less popular item. This extreme skew and difference in request rate between items benefits LFU greatly, especially so when the items are static and thus not subject to change.

5.3.3.2 Weaknesses

LFU's weakness is in its inability to remove items quickly when they no longer experience requests when previously they did. This weakness may result in items staying in the cache for extended periods of time, relative to the amount of requests previously experienced. Examples when LFU may not perform include when the pool of items in the system changes frequently and when floods of requests are experienced for a single item in short succession and then it is no longer requested (e.g. Flash Crowds). For variations of LFU such as Least Frequently Used with Dynamic Aging Least Frequently Used with Dynamic Aging (LFUDA), the issue of over retention of popular items is reduced some, but still may fall pray to flash crowds.

5.4 Bubble Eviction Algorithm

5.4.1 Motivation

LFU has a number of known weaknesses due to the fact that the cache content is kept on a request count basis. Once an item has accumulated a large number of requests, the item is less likely to be dropped by the system. This is a clear issue as items may experience only a short period of popularity before dropping in popularity rapidly as is often experienced in video data, called flash crowds [14, 38, 85]. Another weakness can appear in the form of a Distributed Denial of Service (DDoS) attack which could cause an unusual large request count for videos that are not popular with users of the system, which results in videos remaining in the system cache potentially for an extended period of time even after the DDoS attack as ended (in the case of LFU).

LRU does indeed cope better with flash crowds and DDoS attacks due to the fact it does no retain items for an extended period of time if items do not receive requests after an initial rush. The weakness that can be observed in LRU is the retention of videos which may only have received a single request as it will take a minimum of N requests to expel the video from the cache where N is equal to the size of the cache. A variation of the slowloris (Low-bandwidth) attack [86] may provide a method for polluting a cache with videos not popular with users of the system. LRU considers the most frequently requested item to the most "important" item as it will required a minimum of N requests, where N is equal to the size of the cache, to remove the item from the cache. If unpopular items are inserted into the cache frequently enough, a large portion of the cache with largely be occupied by items rarely requested.

5.4.2 Solution

The newly suggested, simplified algorithm, Bubble, can overcome the weak performance experienced by LRU and LFU without increasing the operation costs. The effectiveness of the algorithm stems from the fact that items, when added to the cache, behave in a stepwise manner which promotes and demotes them in order of popularity. The main

difference between LFU and this new algorithm is that the significance of counting how often items are requested is no longer upheld. Instead the algorithm invests importance in the present popularity of data items to either promote or demote items. This method of caching is beneficial in quick recovery from cache poisoning and rotation of items when individual items experience a rapid decreases and increases (e.g. flood crowds) in popularity when compared to LFU. The items in the cache will change more frequently and will more accurately reflect the present popularity of items. The algorithm discards items swiftly if they experience a sudden decrease in popularity (minimum of N requests where N is equal to the size of the cache).

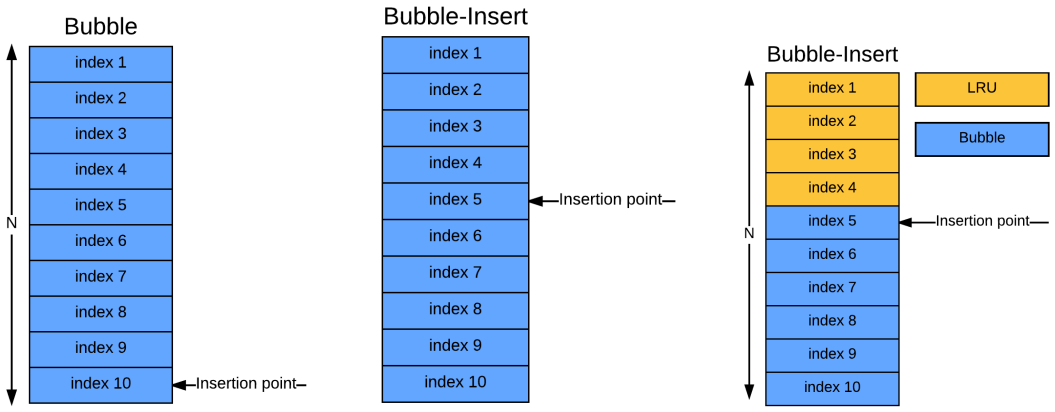


Figure 5.1: Bubble & Bubble Variation Algorithms Diagrams

The Bubble caching algorithm works on a stepwise basis as data items enter the list from the bottom, with each individual request pushing the item towards the top, one space per request, effectively swapping places with the item above the requested item in the list. An example sequence is illustrated in Table 5.1 where we can see a time-line, from left to right, of items introduced into a cache of 3 total spaces. The swapping mechanism can be seen when item 4 experiences a second request in the 6th total request - and again in the 7th total request. The 8th total request sees the introduction of new item 5 which swaps with the item at bottom of the cache, causing the this item to be removed and replaced by item 5, the 5th total request sees item 1 experience a request whilst already at the top of the cache - this will cause no change in the cache.

An example implementation of a caching node with the Bubble caching algorithm

Order of Requested items	1	2	3	4	1	4	4	5	6	6	7	1	2
Cache Content	1	1	1	1	1	1	4	4	4	4	4	4	4
		2	2	2	2	4	1	1	1	6	6	7	7
			3	4	4	2	2	5	6	1	7	1	2

Table 5.1: Bubble Algorithm Example Sequence

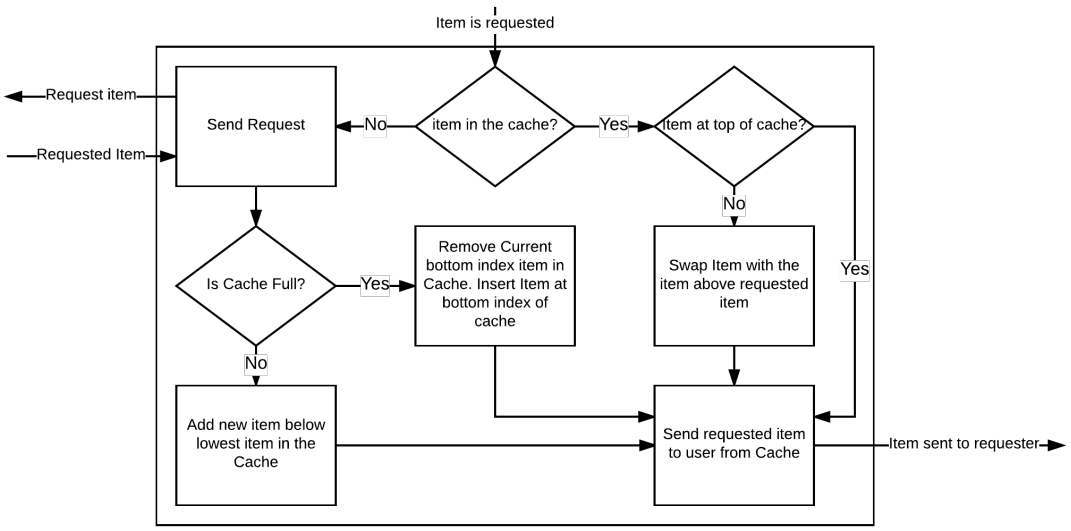


Figure 5.2: Bubble Algorithm Flow Chart

enabled can be found in Figure 5.2.

5.5 Variations of Bubble

Bubble is an algorithm that can be combined with other algorithms. The function of such a cache would best be described as two logically separated caches with one point of insertion with rules governing when items are to be moved between caches.

5.5.1 Bubble-Insert

The Bubble algorithm works by inserting new objects into the bottom of the cache. This means that, for an item to remain in the cache for an extended period of time, it requires multiple requests in a relatively brief time frame. This creates a problem when the skew of item popularities is not steep. The items that would be present in a

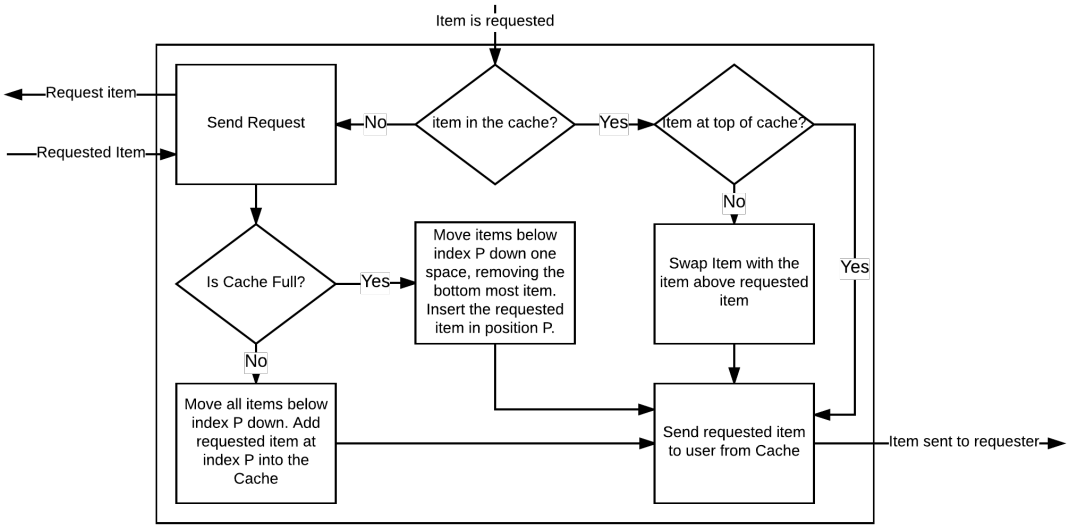


Figure 5.3: Bubble-Insert Algorithm Flow Chart

clairvoyant cache are not submitted into the Bubble cache where they may experience a greater request rate than the current objects inhabiting the cache.

Implementing Bubble with an insertion point in the cache can avoid such issues. The insertion point will function as the new place new items are loaded into the cache, which will cause items below the insertion point to shift down, dropping an item if the cache is currently full. The method of moving down items when a new item is submitted will closely resemble LRU, however it will not be LRU as items below the insertion point will only move up a single space if requested. A flowchart demonstrating how Bubble-Insert functions is shown in Figure 5.3.

The location of insertion can vary and it creates an opportunity to optimise Bubble-Insert. The optimal place for insertion can depend the behavioural characteristics of the data submitted to the cache namely; popularity distribution, item decay rates, item introduction rates.

5.5.2 Bubble-LRU

The Bubble algorithm has a shortcoming in that items that do not receive concurrent requests are not submitted to the cache to be retained for future requests. This remains

true in situations where the quantity of requests for items in the cache is less of those outside the cache. This inherent short-coming prompted research into an improved implementation of Bubble. The Bubble algorithm has the ability of being combined with the LRU cache eviction algorithm which may prove to alleviate the issue of retention of less popular items.
As is true for Bubble-Insert, this solution creates a method of storing and retaining items ensuring a greater turnover of items, expelling items that are thought to provide fewer cache hits than the newly introduced items.

Bubble-LRU separates the cache into two parts that function using different rules. The top most part is the Bubble section which will aim to retain the most popular items in the system of implementation. The bottom section of the cache will function using LRU aiming to hold less popular items, expelling items quickly, should they only receive single requests. Requested items not already in the cache will be inserted into the top of the LRU (referred to as the insertion point P) section of the cache, moving down items and discarding items that spill over the total capacity of the cache. Items are introduced into the Bubble section of the cache if a cache-hit is registered for the item at the top of the LRU cache at the moment of cache-hit. The item that was previously the lowest item in the Bubble cache section, moves into the top of the LRU section of the cache. If an item in the LRU section of the cache is requested it will be moved to the top of the LRU section of the cache, moving all other items down to fill the newly opened space where the item was before receiving a request.

This is demonstrated in the flow chart in figure 5.4.

The Bubble-LRU cache suggests a small improvement can be expected over Bubble-Insert in situations where the bottom part of the cache sees frequent introduction of items that only receive single requests. Bubble-Insert does not have this strength as items in the bottom part of the Bubble-Insert cache may only move an item down to finally be rejected if they are directly below the unpopular item (or outside the cache). In Bubble-LRU any item below the unpopular item (including those outside the cache)

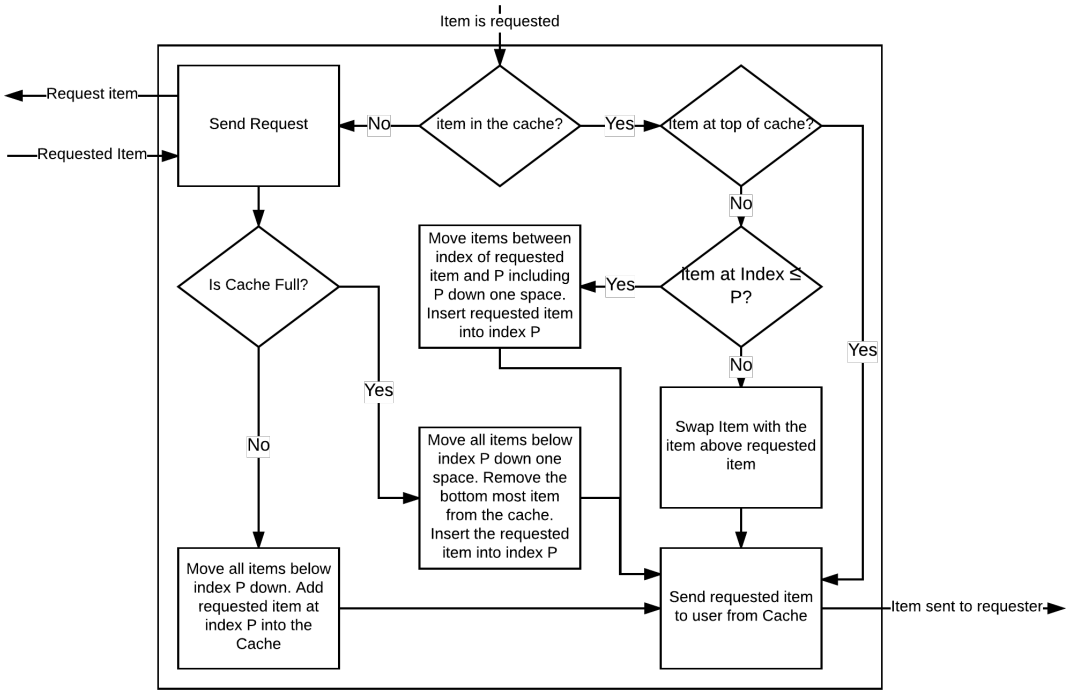


Figure 5.4: Bubble-LRU Algorithm Flow Chart

can move the unpopular item down.

5.6 Introduction to the analytical and simulation techniques

Bubble, Bubble-Insert and Bubble-LRU all provide an alternative to known opportunistic cache eviction algorithms such as LRU, LFU, RAND and FIFO. All the newly introduced algorithms have an operational cost of $O(1)$ just as LRU, RAND and FIFO however a benchmark is required. Benchmarking the newly introduced algorithms against the known algorithms does provide a way of quantifying the usefulness of the newly introduced algorithms, providing a measure of when one of the newly introduced algorithms may prove more efficient than the previously existing ones.

5.6.1 Markov Chain Analysis

Gomaa et al. [87] performs Markov chain analysis on LRU and the FIFO cache eviction algorithms. The analysis presented here will follow the style of Gomaa et al. [87] but for the novel algorithms proposed in this Thesis. The analysis presented by Gomaa et al. [87] presents a novel Markov Chain approach that models the activity of a single web cache during a fixed evaluation interval $[0, T]$. At $t > 0$ the number of objects generated by the web server is $M(t)$ which is considered constant for any t thus $M(t) \approx M$. The requests that arrive at the server are according to a Poisson process with a rate of β. The probability of item requests is according to a Zipf or Zipf-Mandelbrot distribution. The probability of the *ith* item being requested is that of the probability within the Zipf or Zipf-Mandelbrot distribution of the *ith* item multiplied with the request rate (β). The Zipf-Mandelbrot distribution can be generated in combination with the request rate to produce an average request rate for each item λ_i, $1 \leq i \leq M$ as follows:

$$\lambda_i = \frac{\beta}{\sigma(i+\nu)^\alpha}$$

Where $\sigma = \sum_{i=1}^{M} 1/(i+\nu)^\alpha$. In the Zipf-Mandelbrot distribution α represents the steepness of the slope in a Zipf or Zipf-Mandelbrot distribution when plotted on a log-log scale and ν represents the curve of the distribution on a log-log scale. If $\nu = 0$ the distribution is a Zipf distribution.

Each item in the cache has an expected lifetime that is an exponential distribution with mean $1/\mu_i$. It is assumed that α, ν, β and μ_i remain unchanged during the evaluation interval $[0, T]$ across all items.

The cache capacity C is considered as the maximum number of objects, of specified size, that can be stored simultaneously. The assumption is made that all items within the system are of equal size.

The process of estimating the total cache hit ratio of all items $H(t)$ can be performed by first estimating the cache hit ratio of all individual items from 1 to $M, H_i(t)$ and calculating the sum and the probability of request of those items as follows:

$$H(t) = \sum_{i=1}^{M} \frac{H_i(t)}{\sigma(i+\nu)^{\alpha}}$$

$H_i(t)$ is calculated for each item i using the Markov Chain probability matrices P_i^1 as follows:

$$H_i^t = 1 - \sum_{r=1}^{C+1} [P_i^1(t)]_{r,1}$$

If it is assumed that the pool of items in the web server at $t = 0$ is empty, P_i^1 is derived from the flow matrices that is unique to each cache eviction algorithm by getting the exponential as follows: $P_i^1(t) - exp(Q_i \Delta t)$

where $\Delta t = t - t_0.exp(.)$ is an exponential matrix exponential operator.

The superscript in each (as 1 in all examples) refers to the iteration in the iterative algorithm that can be more readily followed in Algorithm 5.1 as variable z which will be used to generate the Exact Cache Hit Ratio. A breakdown of the first iteration of Bubble, Bubble-LRU and Bubble-Insert and the method of calculating the following iterations is detailed in the respective chapters for each eviction algorithm.

The Exact Cache Hit Ratio estimator suggested by [87] requires a probability vector for each item to indicate the probability of location in the cache combined with the probability of request to find the estimated cache hit ratio for each respective item to find the total cache hit ratio for the cache. On the first iteration there is no information yet on the probability of location within the cache of each item, thus an initial estimation is required that does not require the additional probability vector. The additional probability vector is shown to be used in Algorithm 5.1 at the start of the first loop.

LRU and FIFO first have their lower-bound cache hit ratio estimated for the first iteration of this method of finding the Exact Cache Hit Ratio Estimation, however Bubble, Bubble-LRU and Bubble-Insert do not make a lower-bound estimation possible in the method that LRU and FIFO do. Instead, a closer match for the exact cache hit ratio is generated on the first iteration working with the assumption that the item directly below item i in the cache is $\lambda_i - 1$. This is not always true as any item can be in said positions in the cache and receive requests. However, this does give a good estimation for the first iteration.

The process followed in Algorithm 5.1 is the process of finding the Exact Cache Hit Ratio. P_i^z is found by generating the steady state vector from the probability matrix from the iteration previous $(z-1)$ which holds the probability of the location of item i within the cache.

Algorithm 5.1 instantaneous cache hit ratio for any eviction algorithm [87]

procedure ESTIMATING HIT-RATIO
  Calculate: $P_i^1(t) \forall i \in [1, M]$
  Calculate: $H_i^1(t) = 1 - P_(i, 0)^1(t) \forall i \in [1, M]$
  Calculate: $H^1(t)$
  $z = 1$
  repeat
    $z = z + 1$
    Further enhanced estimation of hit ratio with each z increase
    Calculate: $P_i^z(t) using P_i^(z-1)(t) i \in [1, M]$
    Calculate: $H_i^z(t) = 1 - P_(i, 0)^z(t) \forall i \in [1, M]$
    Calculate: $H^z(t)$
  until $(H^z(t) = H^(z-1)(t))$
  Result = $H^z(t)$
end procedure

5.6.1.1 Bubble Implementation

The initial iteration of the iterative algorithm has no prior knowledge of the probability of location of each item, thus the assumption is made that the chance item i moves down in the cache is the chance that the item below item i experiences a request. E.g the chance that the item at the top of the cache is moved down is equal to λ_2 as this item is more probable to be situated in that position within the cache. As the flow

matrix holds the top of the cache at index $C+1$ an index converter was constructed using x, as shown below. For ease of documentation, index 0 will refer to the out of cache state and C as the index of the top of the cache.

$$x = [C, C-1, C-2, , 3, 2, 1]$$

This method of indexing is used for all Bubble cache variations for their initial flow matrix and initial estimation of cache hit ratio.

5.6.1.1.1 First Iteration: inaccurate instantaneous cache hit estimation

The chance of eviction of the item at the bottom of the cache is d_i which is calculated as follows:

$$d_i = \beta - \sum_{k=1,k\neq i}^{C} \lambda_k$$

The flow matrix for Bubble is as follows:

$$Q_i^1 = \begin{bmatrix} -\lambda_i & \lambda_i & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ \mu_i+d_i & -\mu_i-d_i-\lambda_i & \lambda_i & 0 & ... & 0 & 0 & 0 & 0 \\ \mu_i & \lambda_{x[1]} & -\mu_i-\lambda_{x[1]}-\lambda_i & \lambda_i & ... & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & ... & \lambda_{x[C-3]} & -\mu_i-\lambda_{x[C-3]}-\lambda_i & \lambda_i & 0 \\ \mu_i & 0 & 0 & 0 & ... & 0 & \lambda_{x[C-2]} & -\mu_i-\lambda_{x[C-2]}-\lambda_i & \lambda_i \\ \mu_i & 0 & 0 & 0 & ... & 0 & 0 & \lambda_{x[C-1]} & -\mu_i-\lambda_{x[C-1]} \end{bmatrix} \tag{5.1}$$

5.6.1.1.2 Main iteration: Exact instantaneous cache hit estimation

The Exact Cache Hit-ratio for Bubble can be found using the method explained in this chapter. This is done using the same iterative system as Gomaa et al. [87] provide, with an adjusted flow matrix to reflect the movement of items subjected to the Bubble algorithm. Once the flow matrix is generated, the transition matrices, the steady-state vectors and the hit-ratio can be inferred when the process has converged.

The movement for item i from position j to position $j+1$ is λ_i as was for the initial

iteration of Bubble.

The probability vectors from the previous iteration $(z-1)$ are used to find the probability of item i transitioning from state j to state $j-1$ within the cache. For each item $1 \leq k \leq M$ find the probability of it being in the location of $j-1$ combined with the probability of request for each individual item provides the probability of movement from state j to state $j-1$.This can be expressed as follows:

$$\epsilon_{i,j}^{z}(t) = \sum_{k=1,k\neq i}^{M} \lambda_k P_{k,x[j-1]}^{z-1}$$

However, this only expresses an item being in the position below and affecting the item above. In reality, in t, an item other than the one currently in $j-1$ can affect the movement of i in position j. This can be seen as similar to LRU as all items below item i in position j can be any item that sits below j. The probability for item i in position j to $j-1$ in a Bubble enabled cache is as follows:

$$\omega_i = \lambda_i \div \beta$$

For items $1 \leq k \neq i \leq M$ to move i in position j to $j-1$ if $k_j \geq i_j$ is the probability of all items $1 \leq h \neq x \leq M$ in position item k_{j-1} to receive a request multiplied with the probability of h being located in k_{j-1}. This process is repeated $k_j - i_j$ times which provides the probability of item k_j to move to position i_j. The probability of k_j to move to x_{j-1} is:

$$\Omega_{k,j} = \sum_{x=1,x\neq k}^{M} \omega_x$$

For all items where $i_j \geq k_j$, which means for all items below i_j that will influence i_j, the total request rate is:

$$\epsilon_{i,j}^{z}(t) = \sum_{k=1,k\neq i}^{M} \sum_{w=j}^{C} \left(\prod_{h=j}^{w} \Omega_{k,h-1} \right) P_{k,w}^{z-1}$$

For all items where $i_j < k_j$ the total request rate is:

$$\epsilon^z_{i,j}(t) = \sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{1} \left(\prod_{h=(j-1)}^{w} \lambda_h \div \beta \right) P^{z-1}_{k,w}$$

The total sum will encompass all items' probability of being in location i_{j-1} which needs to be multiplies with β to produce the request rate at which i_j moves to i_{j-1}. This is expressed as follows:

$$\epsilon^z_{i,j}(t) = \left(\sum_{k=1,k\neq i}^{M} \sum_{w=j}^{C} \left(\prod_{h=j}^{w} \Omega_{k,h-1} \right) P^{z-1}_{k,w} + \sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{1} \left(\prod_{h=(j-1)}^{w} \lambda_h \div \beta \right) P^{z-1}_{k,w} \right) \beta$$

The probability of item i moving from position j to $j+1$ is λ_i combined with the probability of items $w > j$ experiencing decay μ within the cache is as follows:

$$\gamma^z_{i,j}(t) = \lambda_i + \sum_{k=1,k\neq i}^{M} \mu_k \sum_{w=j-1}^{C} P^{z-1}_{k,w}(t)$$

The flow matrix is as follows:

$$Q^1_i = \begin{bmatrix} -\lambda_i & \lambda_i & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ \mu_i+\epsilon^z_{i,2} & -\mu_i-\epsilon^z_{i,2}-\gamma^z_{i,1} & \gamma^z_{i,1} & 0 & \dots & 0 & 0 & 0 & 0 \\ \mu_i & \epsilon^z_{i,3} & -\mu_i-\epsilon^z_{i,3}-\gamma^z_{i,2} & \gamma^z_{i,2} & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & \dots & \epsilon^z_{i,C-2} & -\mu_i-\epsilon^z_{i,C-2}-\gamma^z_{i,C-1} & \gamma^z_{i,C-1} & 0 \\ \mu_i & 0 & 0 & 0 & \dots & 0 & \epsilon^z_{i,C-1} & -\mu_i-\epsilon^z_{i,C-1}-\gamma^z_{i,C} & \gamma^z_{i,C} \\ \mu_i & 0 & 0 & 0 & \dots & 0 & 0 & \epsilon^z_{i,C} & -\mu_i-\epsilon^z_{i,C} \end{bmatrix} \tag{5.2}$$

5.6.1.2 Bubble-LRU Implementation

Insertion Point P where $1 < P < C$ is an additional variable to consider for Bubble-LRU. P dictates the index insertion of new items into the cache. The items i in state $i_j > P$ are subject to transition only if the item k in state $k_j == i_{j-1}$ receives a request. Items in state $j \leq P$ move to state i_{j-1} if the item k in state $k_j < i_j$ receives a request.

5.6.1.2.1 First Iteration: inaccurate instantaneous cache hit estimation

p_i dictates the rate of transition of states $j \leq P$ to $j-1$ formulated as follows:

$$p_i = \beta - \sum_{k=1,k\neq i}^{P} \lambda_k$$

The transition matrix for the first iteration of Bubble-LRU is therefore as follows:

$$Q_i^1 = \begin{bmatrix} -\lambda_i & 0 & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i+p_i & -\mu_i-p_i-\lambda_i & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i & p_i & -\mu_i-p_i\lambda_i & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & \dots & \lambda_{x[C-3]} & -\mu_i-\lambda_{x[C-3]}-\lambda_i & \lambda_i & 0 \\ \mu_i & 0 & 0 & 0 & \dots & 0 & \lambda_{x[C-2]} & -\mu_i\lambda_{x[C-2]}-\lambda_i & \lambda_i \\ \mu_i & 0 & 0 & 0 & \dots & 0 & 0 & \lambda_{x[C-1]} & -\mu_i-\lambda_{x[C-1]} \end{bmatrix} \tag{5.3}$$

5.6.1.2.2 Main iteration: Exact instantaneous cache hit estimation The transition state i_j to i_{j-1} if $i_j > P$ is dictated by the chance of requests of all items k being moved into position i_{j-1} and receiving a request. The transition from state j to $j-1$ when $j \leq P$ is the probability of any item k in state $P-1$ receiving a request.

The decay of items i in state j where $j \leq P$ in cache Bubble-LRU is $\epsilon_{i,j}^z(t)$ which is formulated as follows:

$$\epsilon_{i,j}^z(t) = \zeta_i - \sum_{k=1,k\neq i}^{M} \lambda_k \sum_{w=j+1}^{C} P_{k,w}^{z-1}(t)$$

The decay experienced by items in position i_j greater than insertion point P is the same as is seen for Bubble in Chapter 5.6.1.1.2 with the additional consideration that items $k_j < P$ influencing i_j to move to i_{j-1} are moved, with a single request, to position P. This requires a small change to the items influencing item i where $x_j < i_j$ (all items k influencing item i in position j that are in positions less than j). This is described as follows:

$$\left(\sum_{k=1,k\neq i}^{M} \sum_{w=(P-1)}^{1} (\lambda_k \div \beta)^{j-P} P_{k,w}^{z-1} \right) \beta$$
$$+ \left(\sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{P} (\lambda_k \div \beta)^{(j-1)-w} P_{k,w}^{z-1} \right) \beta$$

Which provides with the conclusion that $\eta_{i,j}^{z}(t)$, which signifies when items i in state j moves to state $j-1$ for when $j > P$, can be structured as such:

$$\eta_{i,j}^{z}(t) = \left(\sum_{k=1,k\neq i}^{M} \sum_{w=(P-1)}^{1} (\lambda_k \div \beta)^{j-P} P_{k,w}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{P} (\lambda_k \div \beta)^{(j-1)-w} P_{k,w}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=j}^{C} \left(\prod_{h=j}^{w} \Omega_{k,h-1} \right) P_{k,w}^{z-1} \right) \beta$$

The chance of items moving up due to item decay above item i_j is dependent on the state. If state $j \leq P$ the transition from state j to $j+1$ is as follows:

$$\gamma_{i,j}^{z}(t) = \sum_{k=1,k\neq 1}^{M} \mu_k \sum_{w=j+1}^{C} P_{k,w}^{z-1}(t)$$

If state $j \leq P$ the transition from state j to $j+1$ is equal to $\gamma_{i,j}^{z}(t) + \lambda_i$.

p_j will signify the negative values of the rows as j is equal to the column ($x = y$) changes value written as follows:

if $j \leq P$:

$$p_j = -(\mu_i + \epsilon_{i,j}^{z} + \gamma_{i,j}^{z} + \lambda_i)$$

if $j > P$:

$$p_j = -(\mu_i + \eta_{i,j}^{z} + \gamma_{i,j}^{z} + \lambda_i)$$

The flow matrix for Bubble-LRU is as follows:

$$Q_i^1 = \begin{bmatrix} -\lambda_i & 0 & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i + \epsilon_{i,1}^z & p_1 & \gamma_{i,1}^z & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i & \epsilon_{i,2}^z & p_2 & \gamma_{i,2}^z & \dots & \lambda_i & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & \dots & \epsilon_{i,C-2}^z & p_{C-2} & \gamma_{i,C-2}^z + \lambda_i & 0 \\ \mu_i & 0 & 0 & 0 & \dots & 0 & \eta_{i,C-1}^z & p_{C-1} & \gamma_{i,C-1}^z + \lambda_i \\ \mu_i & 0 & 0 & 0 & \dots & 0 & 0 & \eta_{i,C}^z & -\mu_i - \eta_{i,C}^z \end{bmatrix} \tag{5.4}$$

5.6.1.3 Bubble-Insert Implementation

Bubble-Insert functions on the basis that any item from state 0 may be inserted into a specified position P where $1 \leq P \leq C$. This ensure better retention of items newly introduced into the cache than plain Bubble, however functions identically to Bubble in terms of behaviour of items in the cache.

5.6.1.3.1 First Iteration: inaccurate instantaneous cache hit estimation

p_i dictates the rate of transition of states $j \leq P$ to $j-1$ formulated as follows:

$$p_i = \beta - \sum_{k=1,k\neq i}^{C} \lambda_k$$

This is so as only items not in the cache can push items below P towards the bottom of the cache, unlike Bubble-LRU which would enable items in the cache to also be injected into position P if $j < P$

The transition matrix for the first iteration of Bubble-Insert is therefore as follows:

$$Q_i^1 = \begin{bmatrix} -\lambda_i & 0 & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i + p_i & -\mu_i - p_i - \lambda_i & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i & p_i & -\mu_i - p_i \lambda_i & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & \dots & \lambda_{x[C-3]} & -\mu_i - \lambda_{x[C-3]} - \lambda_i & \lambda_i & 0 \\ \mu_i & 0 & 0 & 0 & \dots & 0 & \lambda_{x[C-2]} & -\mu_i \lambda_{x[C-2]} - \lambda_i & \lambda_i \\ \mu_i & 0 & 0 & 0 & \dots & 0 & 0 & \lambda_{x[C-1]} & -\mu_i - \lambda_{x[C-1]} \end{bmatrix} \tag{5.5}$$

5.6.1.3.2 Main iteration: Exact instantaneous cache hit estimation The decay of items $i_j \leq P$ is the rate of request for items k if items $k_j - 1$ receive a request + the probability of items $k_j = o$ receiving a request. As for Bubble and Bubble-LRU, it is important to consider that items not in $i_j - 1$ may receive multiple requests, or may be pushed down to $j - 1$ to move i_j to $i_j - 1$. This is formulated as:

$$\epsilon_{i,j}^{z}(t) = \left(\sum_{k=1,k\neq i}^{M} \lambda_k P_{k,0}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{P} (\lambda_k \div \beta)^{(j-1)-w} P_{k,w}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=j}^{C} \left(\prod_{h=j}^{w} \Omega_{k,h-1} \right) P_{k,w}^{z-1} \right) \beta$$

The decay of items $i_j > P$ is the rate at which items can climb the cache + the rate of items being entered into the cache and moving i_j to $i_j - 1$. This can be expressed as follows:

$$\eta_{i,j}^{z}(t) = \left(\sum_{k=1,k\neq i}^{M} (\lambda_k \div \beta)^{j-P} P_{k,w}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=(j-1)}^{P} (\lambda_k \div \beta)^{(j-1)-w} P_{k,w}^{z-1} \right) \beta + \left(\sum_{k=1,k\neq i}^{M} \sum_{w=j}^{C} \left(\prod_{h=j}^{w} \Omega_{k,h-1} \right) P_{k,w}^{z-1} \right) \beta$$

p_j will signify the negative values of the rows as j is equal to the column ($x = y$) changes value written as follows:

if $j \leq P$:

$$p_j = -(\mu_i + \epsilon_{i,j}^{z} + \gamma_{i,j}^{z} + \lambda_i)$$

if $j > P$:

$$p_j = -(\mu_i + \eta^z_{i,j} + \gamma^z_{i,j} + \lambda_i)$$

The flow matrix for Bubble-Insert is as follows:

$$Q^1_i = \begin{bmatrix} -\lambda_i & 0 & 0 & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i + \epsilon^z_{i,1} & p_1 & \gamma^z_{i,1} & 0 & \dots & \lambda_i & 0 & 0 & 0 \\ \mu_i & \epsilon^z_{i,2} & p_2 & \gamma^z_{i,2} & \dots & \lambda_i & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mu_i & 0 & 0 & 0 & \dots & \epsilon^z_{i,C-2} & p_{C-2} & \gamma^z_{i,C-2} + \lambda_i & 0 \\ \mu_i & 0 & 0 & 0 & \dots & 0 & \eta^z_{i,C-1} & p_{C-1} & \gamma^z_{i,C-1} + \lambda_i \\ \mu_i & 0 & 0 & 0 & \dots & 0 & 0 & \eta^z_{i,C} & -\mu_i - \eta^z_{i,C} \end{bmatrix} \quad (5.6)$$

5.6.1.4 Limitation of analysis of Bubble, Bubble-LRU and Bubble-Insert

The Bubble cache eviction algorithm has a complexity that LRU and FIFO do not have. The primary complexity is that the chance of movement down in the cache can be influenced by any item in the cache, including those above the item in question or many spaces below, which is not true for LRU or FIFO. In a LRU cache, item x below item k influences item k once as multiple request to item x would have the item not influence item k beyond the initial request. In Bubble, a single item below item k can be seen to raise in index in an undisclosed quantity of time to eventually influence item k, which would influenced item k in the initial request in a LRU or FIFO cache. The movement of item x before influencing item k may not be one with a unilateral direction, but rather a number of movements up and down in index before finally influencing k. This complexity presents us with two limitations which are:

1. Any item x may influence the movement of item k_j to move to k_{j-1} by moving down to position $xj = k_{j-1}$ and receiving a request. In order to compute the rate at which x moves down we have to compute the influence of all items that are not x to receive a request whist in the position x_{j-1}. This process may be repeated an

infinite amount of times, increasing the complexity exponentially with each step creating an NP problem.

2. Additionally, the Bubble algorithm can see items k move up and down a number of times before finally moving down item i, which is not accounted for in the suggested Markov-Chain analysis. The reason for this omission are the number of path permutations possible for item k to eventually locate the position below i. The only permutation considered is the probability of an item moving in a single direction (items k below i moving up into position i_{j-1} and items k above item i, and at position i, moving down to position i_{j-1}.

   The inclusion of indirect paths can be expanded infinitely, with each addition of movement to the analysis exponentially increasing the complexity of the calculation of decrease in index of items relative to the size of the cache, thus making this an NP problem

For the reasons above it is declared that only a *first order approximation* is able to be calculated for the results in this Thesis.

5.6.2 Icarus - ICN Simulation Environment

Developed by Saino et al. [43] Icarus is an ICN simulation environment designed to test routing strategies and cache eviction algorithms. The environment enables for a variety of network architectures and environmental factors to be changes to enable benchmarking for specified conditions or a broad spectrum of conditions. For this reason Icarus is well suited to benchmark cache eviction algorithms such as the the proposed Bubble, Bubble-Insert and Bubble-LRU.

Object Popularity Distribution The instance of Icarus used to benchmark the proposed algorithms will have enabled the possibility of implementing a Zipf-Mandelbrot popularity Distribution for the objects available for request as well as a Zipf-like popularity Distribution. This is mainly due to the result of the research conducted in

Chapter 3. It is left to reason that a Zipf-like Distribution for the modelling of popularity data should be considered as well and Zipf-Mandelbrot due to the extensive pre Zipf-Mandelbrot research [7–18] as it is still widely considered a standard regarding what popularity distribution conventional video content follows, as well as general web content which is more specifically considered to follow a pure Zipf distribution [25–27].

Topology The Topology used for the purpose of benchmarking cache eviction algorithms is the GEANT [66] Topology. The purpose of summarising the Icarus results to originate only from a single topology is to reduce the total number of simulations as a greater quantity than already stipulated would require vastly more time. A number of simulations were run including a number of alternative topologies and an extremely minute observable behavioural difference was found to exist between the GEANT topology and the GARR [67],WIDE [69] and Tiscali [68] Topologies - warranting a single topology to be sufficient.

Routing Strategy Routing Strategies are a mayor focus of analysis in Icarus, however it does not encompass the focus of the tests intended for Bubble, Bubble-LRU and Bubble-Insert. For this reason from the large range of routing strategies, a group selected based on diversity will be adopted for the purpose of benchmarking the cache eviction algorithms in question. They are:

- Cache Less For More (CL4M) [48] as in Section 2.3.1
- Leave Copy Down (LCD) [49] as in Section 2.3.2
- Probabilistic Caching (ProbCache) [50] as in Section 2.3.3
- Hash Hybrid Symmetrical-Multicast Routing [51] as in Section 2.3.5.1

Cache Eviction The results observed from the simulation of Bubble, Bubble-LRU and Bubble-Insert will required to be compared to the performance that can be expected from more, well know, cache eviction algorithms that function with an operation cost that is equal or at least similar to Bubble, Bubble-LRU and Bubble-Insert in order to

quantify the realistic performance gain expected, which will be discussed below. This prompts the need for running, what might be considered, a control set of experiments using cache eviction algorithms that will aid in pinpointing the performance gain or loss experienced when Bubble, Bubble-LRU or Bubble-Insert are subjected to Icarus.

The cache eviction algorithms used for setting a benchmark for Bubble, Bubble-LRU and Bubble-Insert to be measured against are:

- Least Recently Used (LRU) as in Section 2.4.1
- Least Frequently Used (LFU) as in Section 2.4.2
- First-In-First-Out (FIFO) as in Section 2.4.4
- Random (RAND) as in Section 2.4.3

5.6.2.1 Icarus Limitations

Icarus does not provide a platform where request objects experience change throughout the simulation. Instead Icarus has provided a static popularity distribution of all request objects which will generate requests only based on the request probability assigned at the start of the simulation. The omission of decay and finite lifespan of items should be noted as a limitation of the Icarus simulation environment when assessing the cache hit ratio of each cache eviction algorithm submitted.

5.6.3 Single Cache Bubble Analysis

The Request Generator used in this section is as introduced in Chapter 4. The request generator provides a tool to simulate requests based on a number of parameters such as: *Probability of Request*, *Decay* (In the form of a function) and *Life-Time*. The parameters used for the simulation were as previously mentioned in the introduction of Chapter 5.7. The function of decay used is an implementation of the function described by Avramova et al. [37]. This decay function is discussed further in Chapter 4.2.1.2. The variables chosen for the purpose of this simulation are: $\tau : 40$ and $\beta : 0.5$.

5.7 Analytical and Simulation Results of Bubble & Variations

The Bubble, Bubble-LRU and Bubble-Insert cache eviction algorithms will be benchmarked using; Markov Chain Analysis, Complex Single Cache Analysis and Icarus. The purpose of these tests is to objectively measure the performance of Bubble, Bubble-LRU and Bubble-Insert against known cache eviction algorithms such as LRU, LFU, FIFO and RAND in a variety of situations relating primarily to web delivery content such as HTML pages, images, videos and other online media objects.

Each method of evaluation suggested and used has pros and cons associated with it. For this reason, multiple methods of testing were used to ensure that, though not each approach is inclusive of all strengths, a plethora of methods with different strengths were utilised. One example is the introduction of object popularity decay and limited lifetime of objects which is included in the method "Complex Single Cache Analysis" and not in the other methods used for testing the cache eviction algorithms.

Each method of testing the cache eviction algorithms will be formulated to reflect the same type of systems with alterations where feasibility of computing may be a limiting factor. The popularity distributions, total number of unique objects, quantity of requests and quantity of requests before observation will be consistent across methods of testing to reflect known online video content characteristics as well as a small spectrum of Zipf-like popularity distributions.

Popularity Distributions as real user request data:

1. Zipf($\alpha = 0.8$) [8]
2. Zipf($\alpha = 1$) [17, 25]
3. Zipf($\alpha = 0.9$) as found in Chapter 3
4. Zipf($\alpha = 0.765$) (motivation stated in Chapter 3)
5. Zipf-Mandelbrot($\alpha = 1.42, \nu = 23$) (motivation stated in Chapter 3)
6. Zipf-Mandelbrot($\alpha = 1.20, \nu = 111$) (motivation stated in Chapter 3)

Popularity Distributions Zipf / Zipf-Mandelbrot:

1. Zipf($\alpha = 0.8$)
2. Zipf($\alpha = 0.9$)
3. Zipf($\alpha = 1$)
4. Zipf($\alpha = 1.1$)
5. Zipf($\alpha = 1.2$)

Total number of unique objects:

1. ≈ 11000
2. ≈ 2500
3. $= 100$ (for the Markov Chain analysis[1])

Requests:

1. Quantity of requests: 600000
2. Quantity of warm-up requests: 300000 (Where applicable)

The real user request distributions will highlight the strengths and weaknesses in pseudo-realistic scenarios. These scenarios are constructed from observed video popularity distributions. They are ranging which suggests that, though they may give an insight into the general performance gained from the suggested algorithms, they are not able to contrast the strengths and weaknesses that Bubble, Bubble-Insert and Bubble-LRU exhibit in Zipf or Zipf-Mandelbrot situations as they are all summarised in the Pseudo-realistic distributions. For this reason a range of Zipf Distributions are also separately tested, as well as a range of Zipf distributions. The Zipf Distributions range from a low α value (0.8) to a larger α value in a stepwise fashion. The Zipf-Mandelbrot distributions used are mapped to be as close to the α variable 0.8 as possible, in an effort to reflect the most realistic Zipf-Mandelbrot distribution with regards to video request behaviour. This was done by increasing the Zipf α variable (steepness of the slope in the Zipf Distribution) and adjusting the skewness factor ν to best reflect the Zipf distribution with α of 0.8. The measurement by which the best Zipf-Mandelbrot distributions were chosen were based on the Kullback-Leibler Divergence results. The

[1]The Markov-Chain analysis is computationally infeasible if attempted with a large pool of unique items; specifically in the case of Bubble, Bubble-Insert and Bubble-LRU.

Model	Optimised Values	KL Divergence
Zipf-Mandelbrot	$\alpha = 1.0, \nu = 5$	0.0429
Zipf-Mandelbrot	$\alpha = 1.2, \nu = 26$	0.114
Zipf-Mandelbrot	$\alpha = 1.4, \nu = 69$	0.177
Zipf-Mandelbrot	$\alpha = 1.6, \nu = 137$	0.226
Zipf-Mandelbrot	$\alpha = 1.8, \nu = 227$	0.265
Zipf-Mandelbrot	$\alpha = 2.0, \nu = 336$	0.295

Table 5.2: Zipf-Mandelbrot correlated to Zipf(a=0.8) with a range of alpha values with fitted ν values confirmed with Kullback-Leiber divergence

results are demonstrated in Table 5.2.

5.7.1 Markov Chain Analysis tests

The Markov Chain analysis has a limitation in the computational power a large scale experiment would require. It is for this reason that a smaller sample of items and cache sizes are used in tests that were performed using the Markov Chain analysis technique developed by Gomaa at al. [87].

As presented in Chapter 5.6.1.4, there are limiting factors in the implementation of Bubble, Bubble-LRU and Bubble-Insert. The limitations are easily observed when running the Markov Chain analysis as the results simply do not converge for these scenarios. This is due to a large cache size that holds a great number of methods for which the decreasing factor for items in the cache to move down can be influenced by other items, as explained in Chapter 5.6.1.4.

The total unique item quantity used in the Markov Chain analysis tests is 100. This is because a larger unique item quantity would require an amount of computational effort that is infeasible in the scope of this PhD.

The tests performed aim to provide a measurement of Bubble, Bubble-LRU and Bubble-Insert's effectiveness in comparison to the effectiveness of LRU,LFU and FIFO. These tests will be structured as follows:

1. What Cache-Hit ratios can be expected from a number of Zipf Distributions with a variety of cache sizes in a system with an item pool of 100 unique items

Bubble-LRU, $M = 100$ Converged Results	Cache Size						
	4	5	6	7	8	9	10
Zipf-M($\alpha = 1.0, \nu = 5$)	3	4	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	2	4	5	5	5	6	5
Zipf-M($\alpha = 1.4, \nu = 69$)	2	2	5	6	6	6	6
Zipf-M($\alpha = 1.6, \nu = 137$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 1.8, \nu = 227$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 2.0, \nu = 336$)	2	2	2	2	2	2	2
Zipf($\alpha = 0.8$)	3	4	5	6	6	6	6
Zipf($\alpha = 0.9$)	3	4	5	6	6	6	7
Zipf($\alpha = 1$)	3	4	5	5	6	6	6
Zipf($\alpha = 1.1$)	3	4	5	6	6	7	7
Zipf($\alpha = 1.2$)	3	4	5	6	6	7	7
Zipf($\alpha = 0.765$)	3	4	5	6	6	6	6
Zipf-M($\alpha = 1.42, \nu = 23$)	2	4	5	5	5	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	2	2	2	2	2	2	2

Table 5.3: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios

2. What Cache-Hit ratios can be expected from a number of Zipf-Mandelbrot Distribution modelled to be similar to a Zipf distribution of α equal to 0.8 (Table 5.2) with a variety of cache sizes in a system with an item pool of 100 unique items.
3. What Cache-Hit ratios can be expected from a number of realistic popularity distributions with a variety of cache sizes in a system with an item pool of 100 unique items.

5.7.1.1 The optimal insertion index for Bubble-LRU and Bubble-Insert

The Bubble-LRU and Bubble-Insert algorithms have a great number of possible indices of insertions, based on the total size of the cache. We first aim to find the most well-performing insertion indices for each set that is to be compared. The sets are comprised of the scenarios presented previously, modelled to give a broad perspective of behaviour of the cache eviction algorithms, as well as a number of pseudo-realistic scenarios.

The results which did converge the insertion indices yielding the highest cache-hit ratios were selected and are displayed in Tables 5.3 & 5.4. The results were derived from sets of results shown in Figure 5.5. The converged result with the insertion index yielding the greatest Hit-Ratio is considered the superior insertion Index for the scenario.

Bubble-Insert, $M = 100$ Converged Results	Cache Size						
	4	5	6	7	8	9	10
Zipf-M($\alpha = 1.0, \nu = 5$)	3	4	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	2	4	5	5	5	6	5
Zipf-M($\alpha = 1.4, \nu = 69$)	2	4	5	6	6	6	5
Zipf-M($\alpha = 1.6, \nu = 137$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 1.8, \nu = 227$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 2.0, \nu = 336$)	2	2	2	2	2	2	2
Zipf($\alpha = 0.8$)	3	4	5	6	6	6	6
Zipf($\alpha = 0.9$)	3	4	5	6	6	6	7
Zipf($\alpha = 1$)	3	4	5	6	6	6	6
Zipf($\alpha = 1.1$)	3	4	5	6	6	7	7
Zipf($\alpha = 1.2$)	3	4	5	6	6	7	7
Zipf($\alpha = 0.765$)	3	4	5	6	6	6	6
Zipf-M($\alpha = 1.42, \nu = 23$)	2	4	5	5	5	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	2	2	2	2	2	2	2

Table 5.4: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios

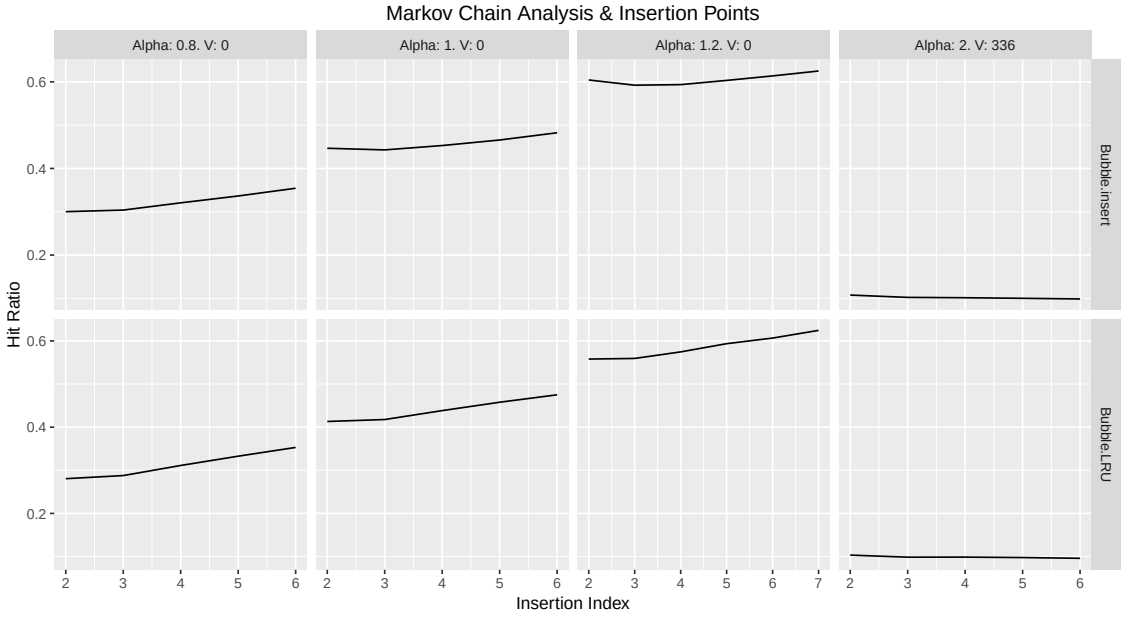


Figure 5.5: Insertion Points - Markov-Chain - Cache Size: 10

A number of Markov-Chain tests did not converge. These results were analysed to show a pattern, though interesting, only show a trend that may suggest that an increased / decreased insertion index is superior, however this suspicion cannot be confirmed using the Markov-Chain analysis performed here due to the Markov-Chain analysis not converging. The analysis results including the non-converged results are shown in Tables

Bubble-LRU, $M = 100$	Cache Size						
Models	4	5	6	7	8	9	10
Zipf-M($\alpha = 1.0, \nu = 5$)	3	4	5	*6	*6-7	*6-8	*6-9
Zipf-M($\alpha = 1.2, \nu = 26$)	2	4	5	*6	*6-7	*7-8	*6-9
Zipf-M($\alpha = 1.4, \nu = 69$)	2	2	5	6	*7	*7-8	*7-9
Zipf-M($\alpha = 1.6, \nu = 137$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 1.8, \nu = 227$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 2.0, \nu = 336$)	2	2	2	2	2	2	2
Zipf($\alpha = 0.8$)	3	4	5	6	*7	*7-8	*7-9
Zipf($\alpha = 0.9$)	3	4	5	6	*7	*7-8	*8-9
Zipf($\alpha = 1$)	3	4	5	*6	*7	*7-8	*7-9
Zipf($\alpha = 1.1$)	3	4	5	6	*7	7	7
Zipf($\alpha = 1.2$)	3	4	5	6	*7	7	7
Zipf($\alpha = 0.765$)	3	4	5	6	*7	*7-8	*7-9
Zipf-M($\alpha = 1.42, \nu = 23$)	2	4	5	*6	*6-7	*6-8	*6-9
Zipf-M($\alpha = 1.20, \nu = 111$)	2	2	2	2	2	2	2

Table 5.5: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios

Insertion Points	Cache Size						
Bubble-Insert, $M = 100$	4	5	6	7	8	9	10
Zipf-M($\alpha = 1.0, \nu = 5$)	3	4	5	*6	*6-7	*6-8	*6-9
Zipf-M($\alpha = 1.2, \nu = 26$)	2	4	5	*6	*6-7	*7-8	*6-9
Zipf-M($\alpha = 1.4, \nu = 69$)	2	4	5	6	*7	*7-8	5
Zipf-M($\alpha = 1.6, \nu = 137$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 1.8, \nu = 227$)	2	2	2	2	2	2	2
Zipf-M($\alpha = 2.0, \nu = 336$)	2	2	2	2	2	2	2
Zipf($\alpha = 0.8$)	3	4	5	6	*7	*8	*8-9
Zipf($\alpha = 0.9$)	3	4	5	6	*7	*7-8	*8-9
Zipf($\alpha = 1$)	3	4	5	6	*7	*7-8	*7-9
Zipf($\alpha = 1.1$)	3	4	5	6	*7	7	7
Zipf($\alpha = 1.2$)	3	4	5	6	*7	7	7
Zipf($\alpha = 0.765$)	3	4	5	6	*7	*7-8	*7-9
Zipf-M($\alpha = 1.42, \nu = 23$)	2	4	5	*6	*6-7	*6-8	*6-9
Zipf-M($\alpha = 1.20, \nu = 111$)	2	2	2	2	2	2	2

Table 5.6: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios

5.5 & 5.6. The results were derived from sets of results shown in Figure 5.6.

The method of selection with regards to the insertion points best suited to represent Bubble-LRU and Bubble-Insert in the scenarios where results did not converge has challenges. The main of which is that for a single cache size, the best performing cache entry point may be one which has not converged (Signified with the * symbol). In order

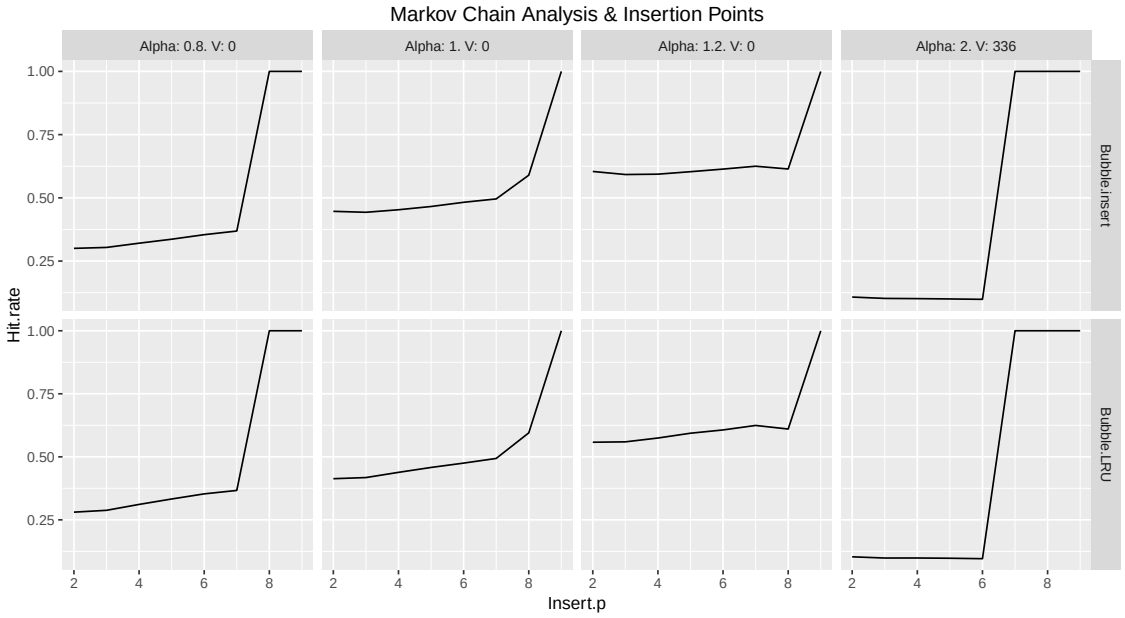


Figure 5.6: Insertion Points - Markov-Chain - Cache Size: 10

to derive a "best" insertion point the way at which the results leaned was taken into consideration. In Figure 5.6 eight example test are shown. In the case of the test where Alpha = 0.8 and V = 0 it is clear to see the best performing insertion point may be one of two points. The converged hit-ratios increases with each incrementation of insertion point until the insertion point have reached 8 or above, which is when an error occurs and the hit-ratio is equal to 1. For this reason the best insertion point is considered 8 or 9. In the case where Alpha = 1.2 and V = 0 it is clear to see that the points that converged insertion point 7 holds the greatest value. Insertion point 9 shows an error by converging to 1, however as insertion point 8 declines from 7 to a lower hit-ratio, it can be assumed that 9 may also follow this trend, thus insertion point 7 is chosen as the representative highest hit-ratio. In the test case where Alpha = 2 and V = 336 we observe that for all values that converged 2 holds the greatest hit-ratio. The hit-ratio declines from 2 as the insertion point increases. For this reason the non-converged errors seen at insertion points 7,8 and 9 are disregarded and the assumption is made the decreasing trend would likely have continued.

5.7.1.2 Markov-Chain Cache-Hit Ratio Results

The constructed scenarios, as listed previously as "Popular Distributions as real request Data" and "Popularity Distributions Zipf / Zipf-Mandelbrot" in the introduction to Section 5.7 will all be subjected to the Bubble, Bubble-LRU and Bubble-Insert algorithms with a varying cache size beside better known cache eviction algorithms such as LRU and FIFO in the case of the Markov-Chain Analysis. The total number of unique items is, as previously stated, 100 with a variety of cache sizes between 4 and 10. For each scenario the Bubble-LRU and Bubble-Insert insertion points have been selected based on their Hit-ratio in each scenario as presented in the previous Chapter. The Bubble-LRU and Bubble-Insert will each be represented once as only the insertion points found to present the best Hit-Ratio will solely represent the algorithm Bubble-LRU or Bubble-Insert in each scenario.

Each group of video popularity distributions described in the introduction of Section 5.7 will be described individually as to address the individual results the cache eviction algorithms produced in each group of tests. All results found in the figures displayed in the following Sections will have the results displayed empirically in Section A.1 of the Appendix.

5.7.1.2.1 Pseudo-Real Video Popularity Distributions, as decided by results found in Chapter 3, were used to aid in measuring the effectiveness of the Bubble, Bubble-LRU and Bubble-Insert cache eviction algorithms, as well as a number of known cache eviction algorithms commonly used such as *LRU* and *FIFO*. The Markov-Chain analysis, as documented in Section 5.6.1, was used to measure the effectiveness of each cache eviction algorithm. The pseudo-realistic distributions considered are listed below. All other simulation details were remain as stated in the introduction of this Section.

1. Zipf($\alpha = 0.8$) [8]
2. Zipf($\alpha = 1$) [17, 25]
3. Zipf($\alpha = 0.9$) as found in Chapter 3
4. Zipf($\alpha = 0.765$) (motivation stated in Chapter 3)

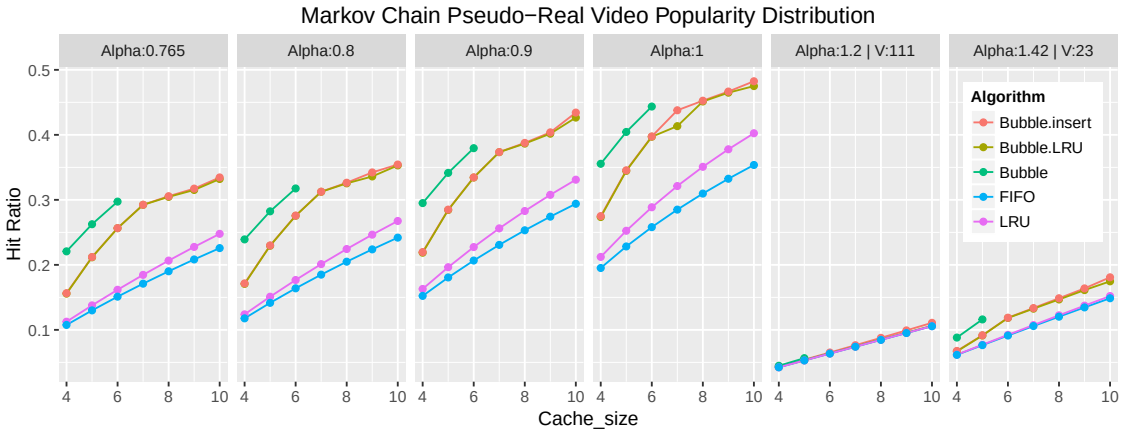


Figure 5.7: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Pseudo-Realistic Video Request Distribution

5. Zipf-Mandelbrot($\alpha = 1.42, \nu = 23$) (motivation stated in Chapter 3)
6. Zipf-Mandelbrot($\alpha = 1.20, \nu = 111$) (motivation stated in Chapter 3)

The results found for each Cache Eviction Algorithm are shown in Figure 5.7. In the Figure, the Cache Size is documented in relation to the Hit Ratio. The Hit Ratios for Bubble, Bubble-LRU and Bubble-Insert in some specific scenarios did not converge due to the limitations of the Markov-Chain Analysis as discussed in Section 5.6.1.4. The results in Figure 5.7 show a clear indication that Bubble, in the case of pseudo-realist where the video popularity distribution follows a Zipf distribution, performs very well if the cache sizes are small. It appears as the ν parameters increase, the Bubble eviction algorithm diminishes in performance. Bubble was unfortunately only converged for small cache sizes which means no conclusion can be drawn as to how effective Bubble would have been for larger cache sizes. Bubble-LRU, as shown in Figure 5.7, shows a Cache-Hit Ratio very similar to Bubble-Insert. The discrepancy between each of the algorithms is very small due to the similar workings of each of the two algorithms. From the test results one may conclude that, for all cache sizes in every pseudo realistic video popularity distribution scenario, Bubble-LRU and Bubble-Insert provide a suitable replacement for the LRU or FIFO algorithms. The operational cost is the same for each of the cache eviction algorithms used with the only additional variable affecting the effectiveness of the Bubble-LRU and Bubble-Insert being the *insertion*

point.

Interestingly, between all pseudo-realistic video request distributions submitted to the Markov-Chain analysis tool, one result does not match the others. The result unlike all other pseudo-realistic test results is the Zipf-Mandelbrot distribution with the parameters $\alpha : 1.2$ and $\nu : 111$. These test results belong to the distribution for which ν is largest and the Zipf-Mandelbrot distribution has the most significant visible curve when plotted on a log-log scale as is visible in Figure 3.9 which shows the VoD distribution to which this Zipf-Mandelbrot distribution was modelled. The most requested items in this distribution are very closely matched, meaning that the likely cause of the low cache-hit ratio is due to the likeness between all top items, exceeding the top 10. This competition between a large amount of the most frequently requested items results in an ineffective cache which is left scrambling between a large set of near equally popular objects, meaning a poor cache-hit ratio is expected.

5.7.1.2.2 Zipf Video Popularity Distributions were tested with increasing α parameters in the Markov-Chain Analysis tool to provide insight into the effectiveness that can be expected when they are submitted to Bubble, Bubble-LRU and Bubble-Insert, as well as a number of known algorithms such as the LRU and FIFO cache eviction algorithms. The increasing α parameters in a Zipf-like distributions submitted to the test provide results that give insight into the eviction algorithms one may prefer in a range of scenarios. The Zipf parameters associated with video delivery are thought to range between $\alpha : 0.8$ [8] and $\alpha : 1$ [17, 25]. Other systems, not exclusively delivering video objects, may follow more extreme power-law distributions thus creating the potential necessity to test algorithms Bubble, Bubble-LRU and Bubble-Insert with more extreme α variables that are greater than 1 as is described in this Section.

The Zipf request distributions submitted to the Markov-Chain analysis are listed below:

1. Zipf($\alpha = 0.8$)
2. Zipf($\alpha = 0.9$)
3. Zipf($\alpha = 1$)
4. Zipf($\alpha = 1.1$)

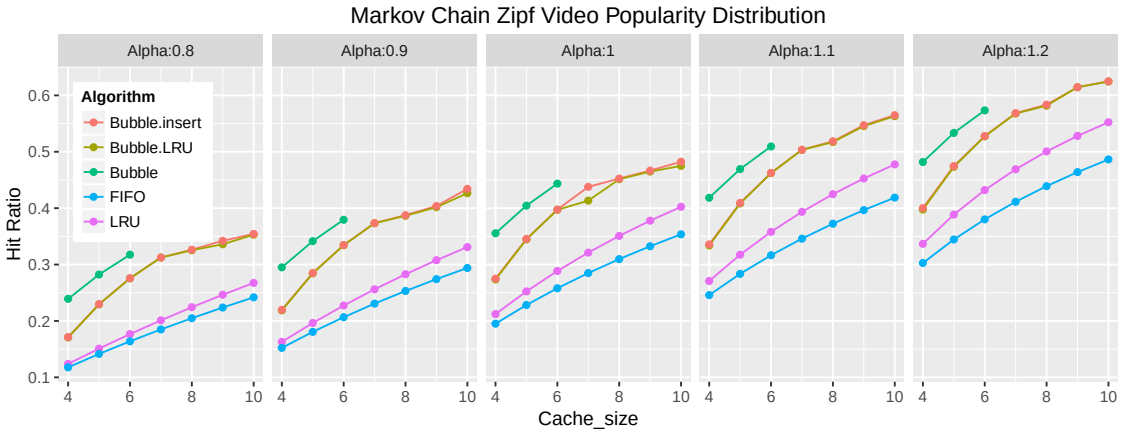


Figure 5.8: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf Video Request Distribution

5. Zipf($\alpha = 1.2$)

From the results displayed in Figure 5.8 a pattern is visible. As the α parameter increases in value, the cache-hit ratio expected from each algorithm can be expected to be higher. The reason for this is that the relative occurrence probability of the most frequently occurring items increases as α increases. Contrasting the Zipf probability $\alpha : 0.8$ to $\alpha : 1.2$; the top 5 items increase from a request probability of 0.319 to a request probability of 0.565 and the top 10 items increase from a request probability of 0.438 to a request probability of 0.685. Acknowledging this insinuates that Bubble, for the few results it did converge, is the most effective cache eviction algorithm. For all cache sizes for which Bubble did not converge Bubble-Insert and Bubble-LRU produced the highest cache-hit ratios. This trend is visible throughout all Zipf-like distributions with little to no deviation as to how difference the cache-hit ratios are in relation to each cache eviction algorithm.

5.7.1.2.3 Zipf-Mandelbrot Video Popularity Distributions modelled to closely approximate the Zipf-like distribution $\alpha : 0.8$ were submitted to the Markov-Chain Analysis method to give insight into the performance one can expect from eviction algorithms Bubble, Bubble-LRU and Bubble-Insert, as well as more commonly know cache eviction algorithms LRU and FIFO, in scenarios with ranging Zipf-Mandelbrot object request distributions. The Zipf-Mandelbrot distributions were modelled to closely approximate

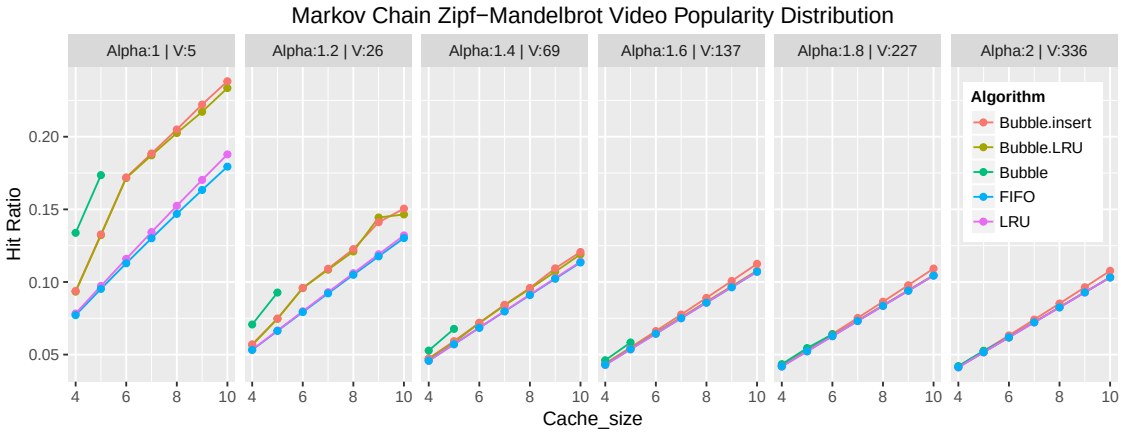


Figure 5.9: Each Figure Displays the results for a different Zipf-Mandelbrot Video Request Distribution

the Zipf-like distribution with parameter $\alpha : 0.8$. α parameter 0.8 was chosen on the basis it is between the Zipf distribution α variables found to most closely approximate the TV catch-up and VoD empirical request distributions as demonstrated in Sections 3.4.2.2 and 3.4.2.1 which uses the KL and PCS comparison methods.

The Zipf-Mandelbrot probability distribution parameters submitted to the Markov-Chain analysis tests to inspect the effectiveness of the aforementioned cache eviction algorithms are listed as follows:

1. Zipf-Mandelbrot$(\alpha = 1.0, \nu = 5)$
2. Zipf-Mandelbrot$(\alpha = 1.2, \nu = 26)$
3. Zipf-Mandelbrot$(\alpha = 1.4, \nu = 69)$
4. Zipf-Mandelbrot$(\alpha = 1.6, \nu = 137)$
5. Zipf-Mandelbrot$(\alpha = 1.8, \nu = 227)$
6. Zipf-Mandelbrot$(\alpha = 2.0, \nu = 336)$

The results of the Markov-Chain analysis testing different Cache-Hit Ratios with probability distributions Zipf-Mandelbrot are shown in Figure 5.9. The Zipf-Mandelbrot distributions are such that, as the α parameter value increases, so does the ν parameter value. As α increases fewer objects draw increasing probability of occurrence. This is counteracted with the parameter ν which creates a more heavy tail in the Zipf-Mandelbrot distribution, as well as a more even request probability among the most

frequently occurring objects. If the large number of frequently requested items share a near equal request probability, the cache eviction algorithms are ineffective in determining which object is more likely to be requested. When this happens, no single algorithm is more effective over another, which may be similarly to a Zipf-Mandelbrot request distribution with a large ν parameter value.

When the ν parameter value is low, the Bubble algorithm appears to be the most effective algorithm for the scenarios were the Bubble eviction algorithm converged. For all other situations, the Bubble-LRU and Bubble-Insert eviction algorithms were the most effective which becomes increasingly less so as the ν parameter increases.

5.7.1.3 Markov-Chain Conclusion

From the results discussed in the previous section it is clearly indicated that, in the case of a static popularity distribution when subjected to the Markov-Chain Analysis method, Bubble, Bubble-LRU and Bubble-Insert are effective cache eviction algorithms, especially relative to the LRU and FIFO cache eviction algorithms.

5.7.2 Icarus Simulations

The Icarus ICN simulation environment created by Saino et al. [51] as further described in Section 2.5 provides a publicly-available, python-based tool for simulating caching behaviour on an ICN network. The popularity distribution of the items populating the system can be changed to reflect the desired system behaviour. Icarus enables the possibility of implementing a number of routing strategies which dictate the flow of data with the intention to populate caches in the network in the most efficient manner thus reducing the total amount of traffic on the network. Icarus also allows for any topology to be implemented. The network topology used for the simulations is GEANT [66] (European academic network). More topologies were initially submitted; however topologies GARR [67], GEANT [66], TISCALI [68] and WIDE [69] produced insignificantly different results and thus, were omitted. The results acquired are measurements of; average cache hit ratio, average path stretch and latency experienced on the network. A num-

ber of popularity metrics will be subjected to testing in regards to request behaviour as described in Section 5.7 however, the simulations will not include details such as; item popularity decay over time, item removal over time and introduction of new items. Instead a static workload will be subjected to testing.

The Routing strategies adopted were "Cache less for more" [48], "Hybrid Symmetrical Hash" [51], "Leave Copy Down" [49] and "Probabilistic Caching" [51] which are representative of the best cache routing strategies implemented in Icarus. The average of all routing strategies makes up the results displayed and presented. The ten cache sizes chosen to test the probability distribution differences for which the range of is displayed in Equation 5.7 where $j = 0, 1, 2 \ldots, 9$ and M is equal to the unique number of video items available for request.

$$\frac{2^j}{1000}M \tag{5.7}$$

It is important to note that the cache capability in each simulation (often referred to as cache size) is relative to the total number of unique items. For example, a cache size of 0.512 for a simulation in which 2500 unique items are available for request will see a total caching capacity of caching capabilities of 1280 which is distributed among all nodes with multiple connections (not edge nodes). This would mean, for the GEANT [66] topology, that the total quantity of caching nodes is 19 which means that a single node is capable of caching 67 unique items. Additional to this, the insertion index for Bubble-LRU and Bubble-Insert is not relative. As the individual cache size for the small group does not reach 5 until a minimum total cache size of 0.064 has been reached and does not reach 5 for the large group until a minimum total cache size of 0.016 has been reached.

Three measurements were obtained from each test with insights into the performance that can be expected from each cache eviction algorithm under a variety of video popularity distributions as described in Section 5.7. The measurements are *Average Latency*, *Average Path Stretch* and *Average Cache Hit-Ratio*.

The number of requests made to the unique items from across edge nodes was 600000

with 300000 warm-up requests made to populate the caches and distribute items in a manner that would reflect a system that has converged to a relatively steady state before the simulation started. The number of requests made should suffice in quantity to ensure a result that are observed have converged to an accurate result, however to ensure consistency, the simulations were performed twice with an average of the two simulations serving as data used in this Chapter.

5.7.2.1 The optimal insertion index for Bubble-LRU and Bubble-Insert

The Bubble-LRU and Bubble-Insert algorithms have a great number of possible indices of insertions, based on the total size of the cache. They were aimed at finding the most well-performing insertion indices for each set that is to be compared. The sets are comprised of the scenarios presented previously modelled to give a broad perspective of behaviour of the cache eviction algorithms, as well as a number of pseudo-realistic scenarios. The optimal insertion point was deduced from the Average Cache Hit-Ratio measurement acquired in the simulations.

The indices found to produce the best results are listed in Tables: 5.7, 5.7, 5.8 and 5.10. Each of the four tables shows a unique set of simulation results for Bubble-Insert and Bubble-LRU for which the unique item count was 2500 or 11000. The best insertion point indices do not appear to follow a steady pattern largely due to the small difference a varying insertion index makes on some distribution of item sets. This small variation of cache hit ratio for a specific set of results can be seen in Figures 5.11 and 5.10. It appears that for a large set of unique item distributions, the insertion index has almost no impact on the cache-hit ratio for when M is large (11000), however when M is small, the insertion index has more impact, likely due to the relative difference between cache size and insertion index. For example Figure 5.10 does not contain the insertion indexes greater than 67 due to the total individual cache size of each node being 67 when the relative cache size is 0.512. When $M = 11000$ a cache enabled node is capable of holding up to 296 when the GEANT topology is applied to the simulation. For this reason the insertion indexes can be expanded to the largest quantity which is 75 for the

Bubble-LRU, $M = 11000$ Results	Relative Cache Size					
	.016	.032	.064	.128	.256	.512
Zipf-M($\alpha = 1.0, \nu = 5$)	5	15	35	70	75	65
Zipf-M($\alpha = 1.2, \nu = 26$)	5	15	35	70	70	65
Zipf-M($\alpha = 1.4, \nu = 69$)	5	15	35	70	75	75
Zipf-M($\alpha = 1.6, \nu = 137$)	5	15	35	70	75	65
Zipf-M($\alpha = 1.8, \nu = 227$)	5	15	35	70	70	75
Zipf-M($\alpha = 2.0, \nu = 336$)	5	15	35	70	70	70
Zipf($\alpha = 0.8$)	5	15	35	65	65	75
Zipf($\alpha = 0.9$)	5	15	35	60	75	70
Zipf($\alpha = 1$)	5	15	35	65	75	50
Zipf($\alpha = 1.1$	5	15	35	60	65	30
Zipf($\alpha = 1.2$)	5	15	35	65	70	15
Zipf($\alpha = 0.765$)	5	15	35	55	75	65
Zipf-M($\alpha = 1.42, \nu = 23$)	5	15	35	70	75	40
Zipf-M($\alpha = 1.20, \nu = 111$)	5	15	35	70	70	70

Table 5.7: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios (Unique Item Count: 11000)

Bubble-Insert, $M = 11000$ Results	Relative Cache Size					
	.016	.032	.064	.128	.256	.512
Zipf-M($\alpha = 1.0, \nu = 5$)	5	15	35	70	75	75
Zipf-M($\alpha = 1.2, \nu = 26$)	5	15	35	70	75	70
Zipf-M($\alpha = 1.4, \nu = 69$)	5	15	35	70	75	70
Zipf-M($\alpha = 1.6, \nu = 137$)	5	15	35	70	75	75
Zipf-M($\alpha = 1.8, \nu = 227$)	5	15	35	70	70	75
Zipf-M($\alpha = 2.0, \nu = 336$)	5	15	35	70	75	70
Zipf($\alpha = 0.8$)	5	15	35	70	70	75
Zipf($\alpha = 0.9$)	5	15	35	65	75	75
Zipf($\alpha = 1$)	5	15	35	70	75	65
Zipf($\alpha = 1.1$)	5	15	35	70	75	70
Zipf($\alpha = 1.2$)	5	15	35	65	70	75
Zipf($\alpha = 0.765$)	5	15	35	70	75	75
Zipf-M($\alpha = 1.42, \nu = 23$)	5	15	35	70	75	75
Zipf-M($\alpha = 1.20, \nu = 111$)	5	15	35	70	75	70

Table 5.8: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios (Unique Item Count: 11000)

tests performed.

The best insertion indexes for both Bubble-LRU and Bubble-Insert are used for the tests in the Section following, in the Icarus test environment, to be representative for their respective cache eviction algorithm.

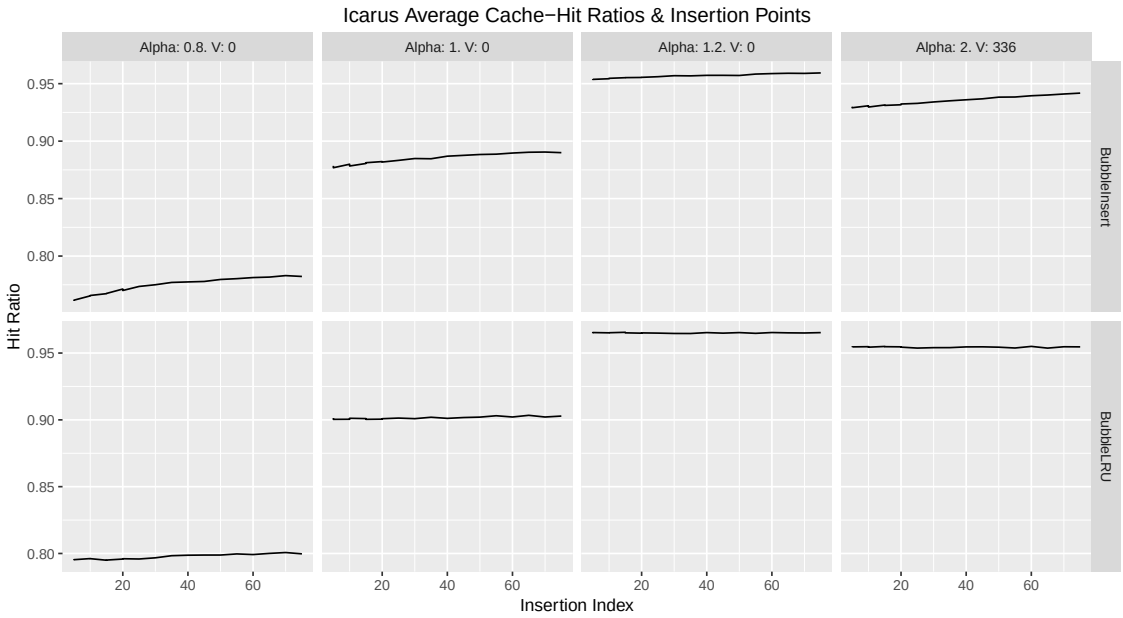


Figure 5.10: Insertion Points - Icarus - Unique Item Count: 11000 - Cache Size: 5635

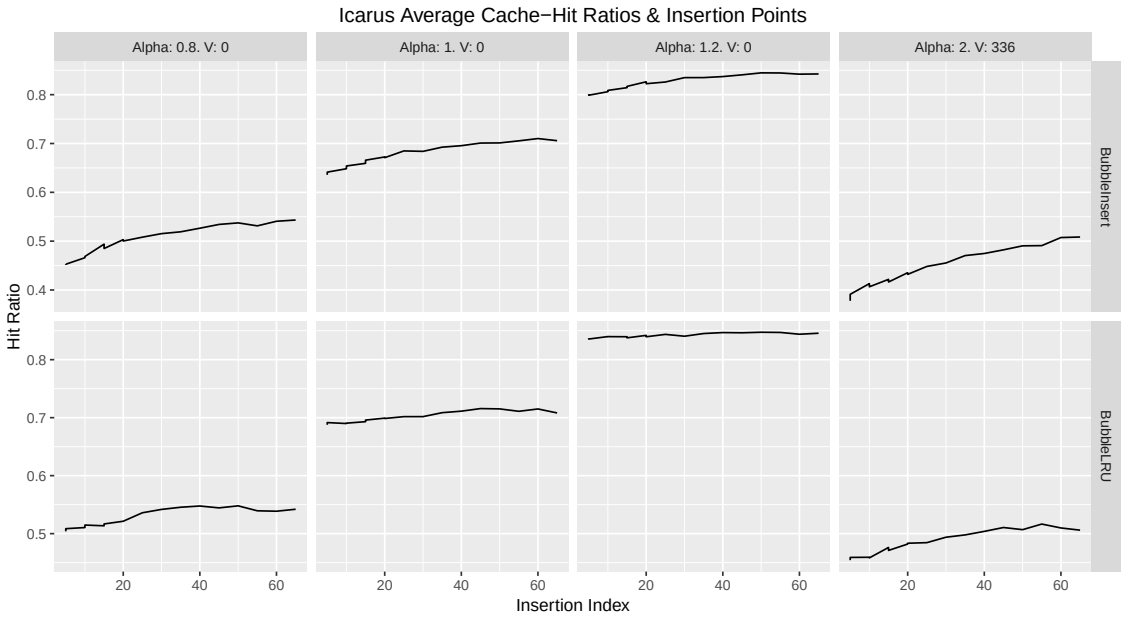


Figure 5.11: Insertion Points - Icarus - Unique Item Count: 2500 - Cache Size: 1280

Bubble-LRU, $M = 2500$ Results	Relative Cache Size			
	.064	.128	.256	.512
Zipf-M($\alpha = 1.0, \nu = 5$)	5	15	30	60
Zipf-M($\alpha = 1.2, \nu = 26$)	5	15	30	60
Zipf-M($\alpha = 1.4, \nu = 69$)	5	15	30	65
Zipf-M($\alpha = 1.6, \nu = 137$)	5	15	30	65
Zipf-M($\alpha = 1.8, \nu = 227$)	5	15	30	65
Zipf-M($\alpha = 2.0, \nu = 336$)	5	15	30	65
Zipf($\alpha = 0.8$)	5	15	30	65
Zipf($\alpha = 0.9$)	5	15	30	65
Zipf($\alpha = 1$)	5	15	30	65
Zipf($\alpha = 1.1$)	5	15	30	55
Zipf($\alpha = 1.2$)	5	15	30	65
Zipf($\alpha = 0.765$)	5	15	30	65
Zipf-M($\alpha = 1.42, \nu = 23$)	5	15	30	65
Zipf-M($\alpha = 1.20, \nu = 111$)	5	15	30	65

Table 5.9: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios (Unique Item Count: 2500)

Bubble-Insert, $M = 2500$ Results	Relative Cache Size			
	.064	.128	.256	.512
Zipf-M($\alpha = 1.0, \nu = 5$)	5	15	30	65
Zipf-M($\alpha = 1.2, \nu = 26$)	5	15	30	65
Zipf-M($\alpha = 1.4, \nu = 69$)	5	15	30	65
Zipf-M($\alpha = 1.6, \nu = 137$)	5	15	30	65
Zipf-M($\alpha = 1.8, \nu = 227$)	5	15	30	65
Zipf-M($\alpha = 2.0, \nu = 336$)	5	15	30	65
Zipf($\alpha = 0.8$)	5	15	30	65
Zipf($\alpha = 0.9$)	5	15	30	65
Zipf($\alpha = 1$)	5	15	30	60
Zipf($\alpha = 1.1$)	5	15	30	65
Zipf($\alpha = 1.2$)	5	15	30	65
Zipf($\alpha = 0.765$)	5	15	30	60
Zipf-M($\alpha = 1.42, \nu = 23$)	5	15	30	65
Zipf-M($\alpha = 1.20, \nu = 111$)	5	15	30	65

Table 5.10: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios (Unique Item Count: 2500)

5.7.2.2 Icarus Cache-Hit Ratio Results

Each group of video popularity distributions described in the introduction of Section 5.7 will be described individually as to address the individual results cache eviction

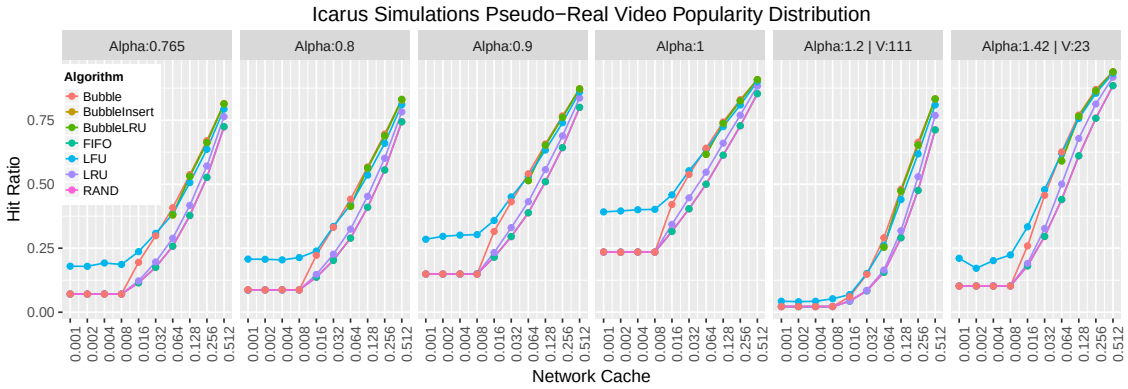


Figure 5.12: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Pseudo-Realistic Video Request Distribution (unique items: 2500)

algorithms produced in each group of simulations.

Pseudo-Real Video Popularity Distributions, as decided by results found in Chapter 3, were used to aid in measuring the effectiveness of the Bubble, Bubble-LRU and Bubble-Insert cache eviction algorithms, as well as a number of known cache eviction algorithms commonly used such as *LRU*, *RAND*, *FIFO* and *FIFO*. The Icarus Simulation, as documented in Section 5.6.2 was used to measure the effectiveness of each cache eviction algorithm. The pseudo-realistic distributions considered are listed below. All other simulation details were as described previously in this Section.

1. Zipf($\alpha = 0.8$) [8]
2. Zipf($\alpha = 1$) [17, 25]
3. Zipf($\alpha = 0.9$) as found in Chapter 3
4. Zipf($\alpha = 0.765$) (motivation stated in Chapter 3)
5. Zipf-Mandelbrot($\alpha = 1.42, \nu = 23$) (motivation stated in Chapter 3)
6. Zipf-Mandelbrot($\alpha = 1.20, \nu = 111$) (motivation stated in Chapter 3)

As can be seen in Figures 5.12 and 5.13, the largest difference in cache-hit-rate can be expected when the cache is of medium size as large caches will be very likely to consistently contain popular items and small caches will frequently discard popular items, regardless of what caching algorithm is used. Table 5.11 shows the mean deviation between Bubble and other known cache eviction algorithms. From the Figures and

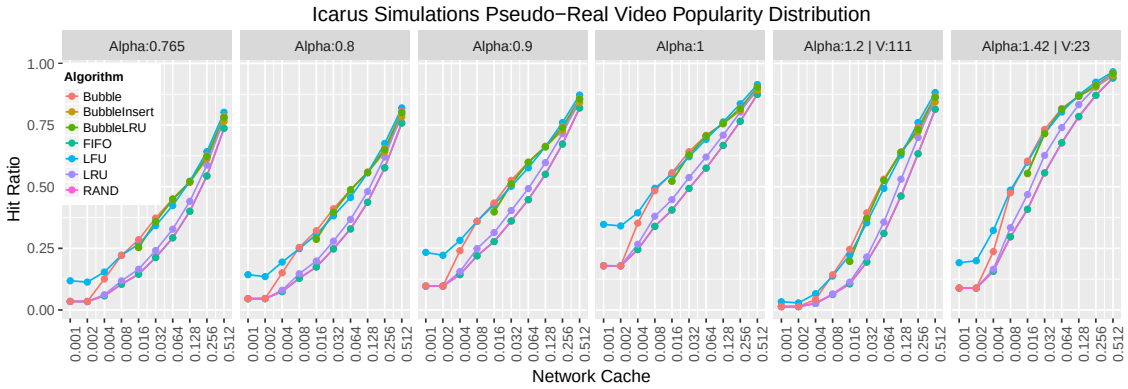


Figure 5.13: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Pseudo-Realistic Video Request Distribution (unique items: 11000)

graphs it is clear that, for the scenarios "pseudo realistic" we can determine Bubble to be an effective cache-eviction algorithm with a total mean performance gain over known cache eviction algorithms of 0.05680574 and −0.02090089 over LFU over all cache sizes. The Bubble algorithm appears to perform best when cache sizes are of a medium size. This is probably due to the repetitive requests required for a single item consecutively in order for an item to be submitted to the cache and remain in the cache for more than a single request cycle. Repetitive requests for single items is likely when the cache size is small due to the power-law nature that is present in the simulation data used.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.001	-0.025	-0.101	0.000	-0.000	0.000
0.002	-0.024	-0.097	-0.000	-0.000	-0.000
0.004	0.042	-0.044	0.074	0.074	0.066
0.008	0.092	-0.002	0.131	0.131	0.108
0.016	0.111	0.012	0.155	0.155	0.123
0.032	0.123	0.027	0.168	0.169	0.129
0.064	0.115	0.025	0.160	0.160	0.115
0.128	0.076	0.001	0.118	0.118	0.070
0.256	0.039	-0.017	0.072	0.072	0.028
0.512	0.018	-0.013	0.040	0.040	0.006

Table 5.11: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms (Unique Item Count: 11000)

As can be seen in Figures 5.12 and 5.13 the largest difference in cache-hit rate can be expected then the cache is of medium size as large caches will, very likely, consistently

contain popular items and small caches will frequently discard popular items, regardless of what caching algorithm is used. Table 5.12 shows the mean deviation between Bubble-Insert and other known cache eviction algorithms. From the Figures and graphs it is clear that, for the scenarios "pseudo realistic" we can determine Bubble-Insert to be an effective cache-eviction algorithm. The Bubble-Insert algorithm appears to perform best when cache sizes are of a medium to large size. This is likely due to the forgiving nature of Bubble-insert to items that do not receive many requests. Items, once submitted, will remain in the cache and can only be expelled by items below them in the cache, receiving requests. Despite this Bubble-Insert would remain the algorithm of choice for the scenarios for which the cache size would be of size 0.016 to 0.126.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.071	-0.028	0.115	0.115	0.083
0.032	0.107	0.011	0.153	0.153	0.113
0.064	0.113	0.022	0.158	0.158	0.112
0.128	0.075	-0.001	0.116	0.116	0.068
0.256	0.026	-0.029	0.060	0.060	0.015
0.512	0.000	-0.031	0.021	0.021	-0.012

Table 5.12: Mean difference of the cache-hit ratios between Bubble-Insert and other Cache Eviction Algorithms (Unique Item Count: 11000)

As can be seen in Figures 5.12 and 5.13 the largest difference in cache-hit-rate can be expected when the cache is of medium size as large caches will be very likely to consistently contain popular items and small caches will frequently discard popular items, regardless of what caching algorithm is used. Table 5.13 shows the mean deviation between Bubble-LRU and other known cache eviction algorithms. From Figures 5.12 and 5.13 and Table table: Mean Deviation pseudo real BubbleLRU, it is clear that for the scenarios "pseudo realistic" we can determine Bubble-LRU to be an effective cache-eviction algorithm. The Bubble-LRU algorithm appears to perform best when cache sizes are of a medium and large size. This can be expected due to the quick removal of items that are rarely requested in the LRU section of the cache and the retention of items that are frequently requested within the bubble section of the cache.

To conclude: It is important to remember that when the insertion index of Bubble-

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.073	-0.026	0.116	0.116	0.085
0.032	0.107	0.011	0.153	0.153	0.113
0.064	0.113	0.023	0.158	0.158	0.113
0.128	0.075	-0.000	0.117	0.117	0.069
0.256	0.035	-0.021	0.068	0.068	0.024
0.512	0.015	-0.016	0.036	0.036	0.003

Table 5.13: Mean difference of the cache-hit ratios between Bubble-LRU and other Cache Eviction Algorithms (Unique Item Count: 11000)

Insert and Bubble-LRU is at the lowest index possible of the cache, it is the Bubble algorithm. For this reason, is it interesting to observe that the Bubble algorithm holds a high cache-hit ratio when: the cache sizes are of a medium size, in the case of a large set of unique available items for request; or, there is a medium to large cache size in the case of a smaller set of unique items available for request. This is likely due to the larger set holding the most frequently requested items in a smaller amount of items relatively to the total size of the cache (e.g. In the case of pure Zipf, if the total number of unique available items is 2500, the cumulative request probability of the 1% of items, receive 46% of the total requests.

The probability of 1% of a set of 11000 unique items following a Zipf distribution is 53%). The situation in which Bubble, and it's counterparts Bubble-LRU and Bubble-Insert, can be considered most effective is when the number most frequently requested items share the same size as the cache when the cumulative request probability of those items is x, which is a variant that depends on the request probability distribution. Based on the average of all results for set sizes 11000 and 2500 in the scenario of Pseudo realistic request distributions when the count of the most frequently requested items that have a combined request probability of approximately 70% is equal to the total network cache size it is likely to see Bubble perform better than other algorithms listed in the simulations performed in Icarus. This result will vary depending on the probability distribution found to exist in an environment. As a closing remark, it is interesting to observe that LFU has the largest operational cost out of all the other algorithms compared in this set of simulation results, meaning that Bubble is the most effective

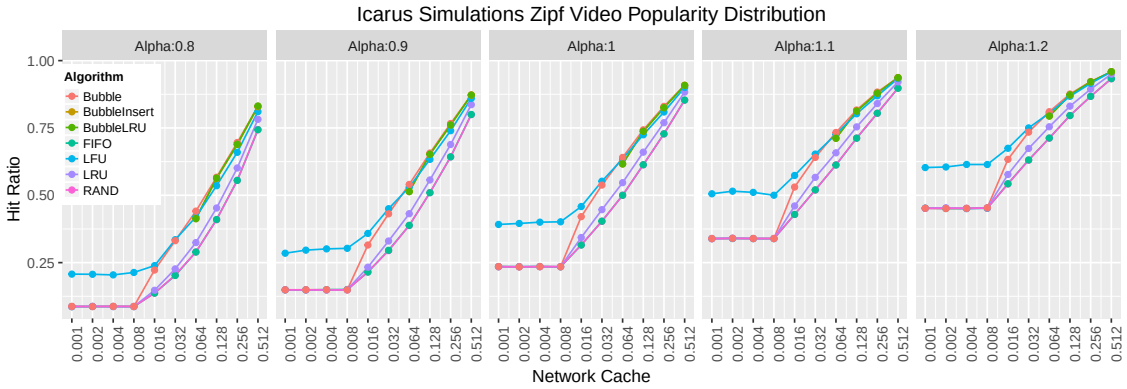


Figure 5.14: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf Video Request Distribution (unique items: 2500)

algorithm, in all instances, with an operational cost of O(n).

Zipf Video Popularity Distributions were tested with increasing α parameters in the Markov-Chain Analysis tool to provide insight into the effectiveness that can be expected when they are submitted to Bubble, Bubble-LRU and Bubble-Insert, as well as a number of known algorithms such as the LRU and FIFO cache eviction algorithms. The increasing α parameters in a Zipf-like distribution submitted to the test provide results that give insight into the eviction algorithm one may prefer that looked at a range of scenarios. The Zipf parameters associated with video delivery are thought to range between $\alpha : 0.8$ [8] and $\alpha : 1$ [17, 25]. Other systems, not exclusively delivering video objects, may follow more extreme power-law distributions thus creating the potential necessity to test algorithms Bubble, Bubble-LRU and Bubble-Insert with more extreme α variables that are greater than 1, as described in this Section.

The Zipf request distributions submitted to the Markov-Chain analysis are listed below:

1. Zipf($\alpha = 0.8$)
2. Zipf($\alpha = 0.9$)
3. Zipf($\alpha = 1$)
4. Zipf($\alpha = 1.1$)
5. Zipf($\alpha = 1.2$)

Figures 5.15 and 5.14 demonstrate the cache-hit rates for the cache eviction algo-

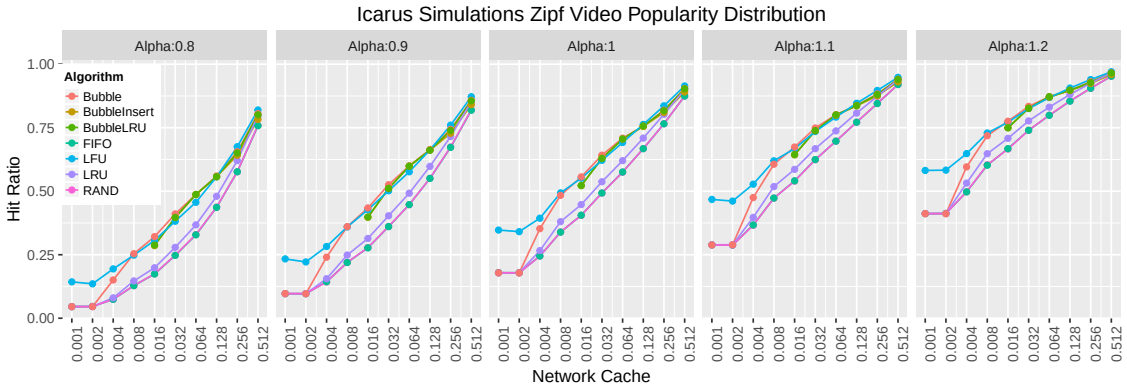


Figure 5.15: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf Video Request Distribution (unique items: 11000)

rithms listed in the legends in each of the figures. The graphs suggest that LFU, Bubble, Bubble-LRU and Bubble-Insert are predominantly the most efficient algorithms. To further emphasise this, a number of tables were included; (Tables 5.14, 5.15 and 5.16). The tables demonstrate the different cache-hit ratios of all algorithms with all popularity distribution used in all simulations combined.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.001	-0.038	-0.150	-0.000	-0.000	-0.000
0.002	-0.036	-0.144	0.000	-0.000	-0.000
0.004	0.056	-0.046	0.098	0.097	0.077
0.008	0.089	-0.005	0.132	0.132	0.096
0.016	0.097	0.007	0.139	0.139	0.101
0.032	0.099	0.019	0.139	0.139	0.099
0.064	0.087	0.017	0.124	0.124	0.084
0.128	0.054	-0.004	0.086	0.087	0.048
0.256	0.028	-0.015	0.053	0.054	0.019
0.512	0.013	-0.010	0.030	0.030	0.004

Table 5.14: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

Bubble as seen in Figures 5.15 and 5.14, performs well in most simulations. From the Figures it is hard to distinguish the difference in performance between the Bubble algorithm and LFU, Bubble-LRU and Bubble-Insert algorithms in all probability distributions shown. Table 5.14 demonstrates the mean difference that is observed when all

Zipf-like distributions are combined and averaged to assess how Bubble compares to all other known algorithms used in the simulations. It is easy to see that Bubble, although not always superior over LFU in all situations, does appear to, in some instances, perform better, and achieve a greater cache-hit ratio, than its better known counterpart. The performance gain over LFU can be considered small when it is achieved, however, it is often substantial over the cache eviction algorithms that share an equal operational cost of O(n). The total mean difference observed for Bubble in this set of simulations is 0.045.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.065	-0.024	0.108	0.108	0.070
0.032	0.087	0.007	0.127	0.127	0.088
0.064	0.086	0.016	0.123	0.123	0.083
0.128	0.053	-0.005	0.085	0.086	0.047
0.256	0.024	-0.018	0.050	0.050	0.016
0.512	0.012	-0.012	0.028	0.028	0.002

Table 5.15: Mean difference of the cache-hit ratios between Bubble-LRU and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

Bubble-LRU as seen in Figures 5.15 and 5.14 performs well in most simulations. From the Figures, it is hard to distinguish the difference in performance between the Bubble-LRU algorithm and LFU, Bubble and Bubble-Insert algorithms in all probability distributions shown. Table 5.15 demonstrates the mean difference that is observed when all Zipf-like distributions are combined and averaged to assess how Bubble-LRU compares to all other known algorithms used in the simulations. From this set of observations, it is likely that Bubble-LRU would not be considered over LFU as the situations for which Bubble-LRU is better, is only better by a very slight margin.

Bubble-Insert as seen in Figures 5.15 and 5.14 performs well in most simulations. From the Figures it is hard to distinguish the difference in performance between the Bubble-Insert algorithm and LFU, Bubble and Bubble-LRU algorithms in all probability distributions shown. Table 5.16 demonstrates the mean difference that is observed when all Zipf-like distributions are combined and averaged to assess how Bubble-Insert compares to all other known algorithms used in the simulations. From this set of

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.064	-0.025	0.107	0.106	0.069
0.032	0.088	0.007	0.127	0.127	0.088
0.064	0.086	0.015	0.122	0.123	0.082
0.128	0.053	-0.005	0.085	0.085	0.046
0.256	0.017	-0.026	0.042	0.043	0.008
0.512	-0.001	-0.024	0.015	0.016	-0.010

Table 5.16: Mean difference of the cache-hit ratios between Bubble-Insert and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

observation, it can be seen that the results for Bubble-LRU and Bubble-Insert are very comparable, meaning that LFU would be the preferred algorithm in almost all cases.

To conclude: Reflecting on Tables 5.14, 5.15 and 5.16 it can be determined that Bubble is the best newly introduced cache eviction algorithm. However, being the best newly introduced algorithm is situational. The statement of being the best algorithm comes from the comparison to all the algorithms that were simulated as part of the Zipf results. Bubble does indeed appear to be the preferred algorithm in cache sizes that are of a medium size. The results found for the Zipf distributions tested appears to align with the results in the previous Section in which the pseudo-realistic probability distributions were applied in the Icarus simulations. The situation in which Bubble can be considered most effective is when the number of most frequently requested items share the same size as the cache when the cumulative request probability of those items is x which is a variant depending on the request probability distribution. Based on the average of all results for set sizes 11000 and 2500 in the scenario of Pseudo realistic request distributions when the count of the most frequently requested items that have a combined request probability of approximately 70% is equal to the total network cache size we are likely to to see Bubble perform better than other algorithms listen in the simulations performed in Icarus.

Zipf-Mandelbrot Video Popularity Distributions modelled to closely approximate the Zipf-like distribution $\alpha : 0.8$ were submitted to the Markov-Chain Analysis method to give insight into the performance one can expect from eviction algorithms

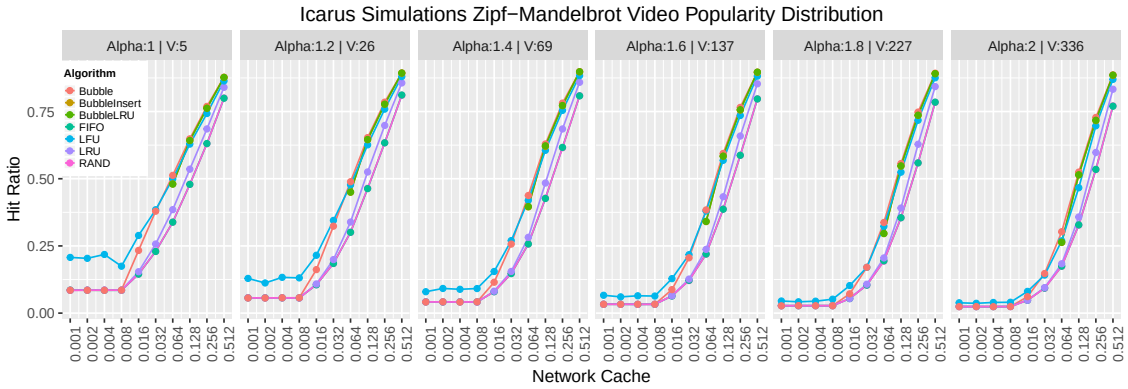


Figure 5.16: Each Figure Displays the results for a different Zipf-Mandelbrot Video Request Distribution (unique items: 2500)

Bubble, Bubble-LRU and Bubble-Insert, as well as more commonly know cache eviction algorithms LRU and FIFO, in scenarios with ranging Zipf-Mandelbrot object request distributions. The Zipf-Mandelbrot distributions were modelled to closely approximate the Zipf-like distribution with parameter $\alpha = 0.8$. α parameter 0.8 was chosen on the basis that it is between the Zipf distribution α variables found to most closely approximate the TV catch-up and VoD empirical request distributions as demonstrated in Sections 3.4.2.2 and 3.4.2.1 which uses the KL and PCS comparison methods.

The Zipf-Mandelbrot probability distribution parameters submitted to the Markov-Chain analysis tests to inspect the effectiveness of the aforementioned cache eviction algorithms and are listed as follows:

1. Zipf-Mandelbrot($\alpha = 1.0, \nu = 5$)
2. Zipf-Mandelbrot($\alpha = 1.2, \nu = 26$)
3. Zipf-Mandelbrot($\alpha = 1.4, \nu = 69$)
4. Zipf-Mandelbrot($\alpha = 1.6, \nu = 137$)
5. Zipf-Mandelbrot($\alpha = 1.8, \nu = 227$)
6. Zipf-Mandelbrot($\alpha = 2.0, \nu = 336$)

Figures 5.17 and 5.16 demonstrate the cache-hit rates for the cache eviction algorithms listed in the legends in each of the figures. The graphs suggest that LFU, Bubble, Bubble-LRU and Bubble-Insert are predominantly the most efficient algorithms. To further emphasise this, a number of tables were included (Tables 5.17, 5.18 and 5.19). The

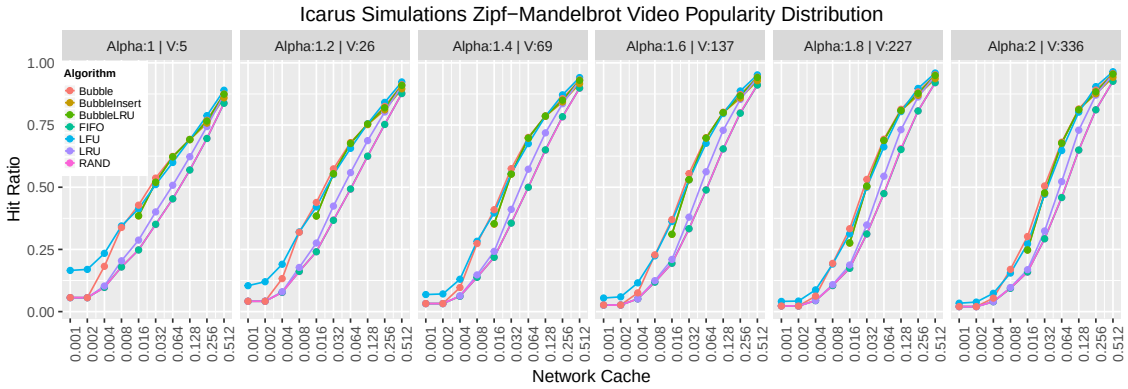


Figure 5.17: Each Figure Displays the results for a different Zipf-Mandelbrot Video Request Distribution (unique items: 11000)

tables demonstrate the different cache-hit ratios of all algorithms with all popularity distribution used in all simulations combined.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.001	-0.011	-0.045	0.000	0.000	0.000
0.002	-0.013	-0.050	0.000	0.000	-0.000
0.004	0.019	-0.038	0.039	0.039	0.037
0.008	0.088	0.001	0.121	0.121	0.111
0.016	0.130	0.017	0.175	0.175	0.152
0.032	0.153	0.027	0.211	0.211	0.165
0.064	0.141	0.027	0.201	0.202	0.135
0.128	0.091	0.005	0.144	0.144	0.074
0.256	0.039	-0.015	0.075	0.075	0.022
0.512	0.015	-0.010	0.033	0.033	0.003

Table 5.17: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

The Bubble algorithm can be considered the best able to contain frequently requested items in a number of scenarios displayed in Figures 5.17 and 5.16 which can be more clearly seen in Table 5.17. The results conclude that LFU, in most tested scenarios, performs better than Bubble, however not when the cache size is of a medium size. Considering only cache eviction algorithms that are of an equal operational cost (LRU, FIFO and RAND) it is interesting to see that Bubble has a much superior cache-hit ratio. For these reasons Bubble should be considered. As for the majority of scenarios tested, the mean difference between Bubble and all other cache eviction algorithms

for the scenarios Zipf-Mandelbrot was positive, meaning that the mean cache-hit ratio was above average, and in some occasions better than all other mean cache eviction algorithms used in the Icarus simulations.

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.076	-0.036	0.121	0.121	0.098
0.032	0.130	0.004	0.188	0.188	0.142
0.064	0.138	0.024	0.198	0.198	0.132
0.128	0.090	0.003	0.142	0.142	0.073
0.256	0.034	-0.021	0.069	0.070	0.016
0.512	0.013	-0.012	0.031	0.031	0.001

Table 5.18: Mean difference of the cache-hit ratios between Bubble-LRU and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

Cache Size	Mean Difference	LFU	RAND	FIFO	LRU
0.016	0.075	-0.038	0.120	0.120	0.097
0.032	0.130	0.003	0.187	0.187	0.141
0.064	0.138	0.024	0.199	0.199	0.132
0.128	0.089	0.002	0.141	0.141	0.072
0.256	0.025	-0.029	0.061	0.061	0.008
0.512	-0.001	-0.025	0.018	0.018	-0.013

Table 5.19: Mean difference of the cache-hit ratios between Bubble-Insert and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 11000)

Bubble-LRU and Bubble-Insert in the scenarios of Zipf-Mandelbrot are of a very comparable performance. Each algorithm demonstrates positive results when cache sizes are of a medium size within the ranges simulated. When cache sizes are of a favourable size to Bubble-Insert and Bubble-LRU, it can be seen that the performance over the next best algorithm, LFU, is rather small relative to the performance LFU provides over Bubble-LRU and Bubble-Insert in the rest of the simulation results. In a few cases, Bubble-LRU and Bubble-Insert have a negative mean difference when compared to all algorithms in the simulation, which makes it hard to value these algorithms for the simulation in which Zipf-Mandelbrot was applied. If the Tables of Mean Difference of Bubble-Insert and Bubble-LRU are compared to the Table of Mean Difference of Bubble and it can be concluded that Bubble is the superior algorithm out of those

newly introduced.

5.7.2.3 Bubble, Bubble-insert and Bubble-LRU Icarus Simulation Conclusion

Bubble, Bubble-LRU and Bubble-Insert are, in this chapter, compared against algorithms of similar or greater operational costs in a video on demand setting. A video on demand system may show a variety of characteristics, thus presenting the need to test a number of them to identify how the presented algorithms may perform under different circumstances. When summarising the results in the Zipf, Zipf-Mandelbrot and Pseudo realistic simulations it is clear to see that LFU and Bubble are very comparable in performance when cache sizes range between 0.008 and 0.128. In these scenarios one can expect Bubble to produce a cache-hit ratio comparable, or better, than LFU. If only cache eviction algorithms on operational cost $O(n)$ were considered, Bubble would certainly be the most effective algorithm in the scenarios tests.

This concludes the Icarus simulation discussion with a situational preference for Bubble over other algorithms used in this chapter. Bubble-Insert and Bubble-LRU are tested, however perform to a mediocre standard when compared to Bubble with a lower cache-hit ratio across all cache sizes. Bubble-Insert and Bubble-LRU do provide an appropriate alternative to algorithms tested with an operational cost of O(n), however, they are more cumbersome to set-up correctly than Bubble with an additional variable that is required to be set accordingly to gain the performance witnessed in the simulations in this chapter (Insertion Index).

5.7.3 Complex Request Single Cache Analysis

Objects in real systems do not exist with a constant request frequency throughout the period they are available for request. The majority of objects experience a measure of popularity when they are first introduced into the system with a gradual decay of popularity in some web systems, such as video delivery systems [37]. To simulate this, it is important to consider the churn one may expect of videos in a VoD system which can

be variable depending largely on the category of content [39] (e.g. News, Movies, TV shows). Icarus and the Markov Chain Analysis used for testing the Bubble algorithms as well as its variations do not encompass this aspect of video delivery systems. The Complex Request Single Cache Analysis simulator does encompass this aspect of video delivery for which there are two parts. One is the Request Generator and the second is a single cache to which the requests are subjected.

The Request Generator used in this section is as introduced in chapter 4. The request generator provides a tool to simulate requests based on a number of parameters such as: *Probability of Request*, *Decay* (In the form of a function) and *Life-Time*. The parameters used for the simulation were as previously mentioned in the introduction of Chapter 5.7. The function of decay used is an implementation of the function described by Avramova et al. [37]. This decay function is discussed further in Chapter 4.2.1.2. The variables chosen for the purpose of this simulation are:$\tau : 40$ and $\beta : 0.5$.

The tests performed aim to provide a measurement of Bubble, Bubble-LRU and Bubble-Insert's effectiveness in comparison to the effectiveness of LRU,LFU and FIFO. these tests will be structured as follows:

1. What Cache-Hit ratios can be expected from a number of Zipf Distributions with a variety of cache sizes in a system with an item pool of 11000 or 2500 unique items
2. What Cache-Hit ratios can be expected from a number of Zipf-Mandelbrot Distribution modelled to be similar to a Zipf distribution of α equal to 0.8 (Table 5.2) with a variety of cache sizes in a system with an item pool of 11000 or 2500 unique items.
3. What Cache-Hit ratios can be expected from a number of pseudo realistic popularity distributions with a variety of cache sizes in a system with an item pool of 11000 or 2500 unique items.

Note the cache sizes used in the tests performed in this chapter are deliberately sized to be comparable to the Icarus simulation results. The relative cache sizes are $S = [0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256, 0.512]$ to the total cache

thus, making the cache sizes used $M \times S$. For when $M = 2500$, the smallest two values are dismissed, due to their extremely small size ($[2, 5]$) when measuring Bubble-LRU and Bubble-Insert as the lowest insertion point is 5.

5.7.3.1 The optimal insertion index for Bubble-LRU and Bubble-Insert

The Bubble-LRU and Bubble-Insert algorithms have a great number of possible indices of insertions, based on the total size of the cache. The aim is to find the most well-performing insertion indices for each set that is to be compared. The sets are comprised of the scenarios presented previously, modelled to give a broad perspective of behaviour of the cache eviction algorithms, as well as a number of pseudo-realistic scenarios. For the Complex Request Single Cache Analysis scenario, Bubble-LRU and Bubble-Insert can be assessed in two scenarios in which the total available pool of items is small and large ($M = 11000$ OR $M = 2500$).

5.7.3.1.1 The large dataset where $M = 11000$ is interesting when looked at separately from the smaller dataset due to the rotational influence that is introduced in this Chapter. The larger dataset M can influence the cache behaviour differently from when M is small through the larger set of frequently requested items. similar decay and removal rate relative to M would suggest a difference in request behaviours between the large and small item set M.

Tables 5.21 and 5.20 demonstrate the changing preference in insertion index dependant on the frequency of request distributions submitted to the caching environment. The insertion index is largely preferred to be low with an index of 5 with the exception of when the popularity distribution is Zipf $\alpha = 1$ and larger. The low insertion index is likely due to the requirement that items do not remain in the cache for a long period of time. Once submitted to the cache an item will decay rapidly in their frequency of requests. This characteristic of the Complex Request Single Cache Analysis gives a unique insight that may be visible in terms on best insertion index.

Bubble-LRU, $M = 11000$ Converged Results	Cache Size									
	11	22	44	88	176	352	704	1408	2816	5632
Zipf-M($\alpha = 1.0, \nu = 5$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	5	5	5	5	5	5	10	10	5	5
Zipf-M($\alpha = 1.4, \nu = 69$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.6, \nu = 137$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.8, \nu = 227$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 2.0, \nu = 336$)	5	5	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.8$)	5	5	5	5	5	5	10	5	5	5
Zipf($\alpha = 0.9$)	5	5	5	5	5	5	15	5	10	5
Zipf($\alpha = 1.0$)	10	10	10	10	10	10	20	15	15	5
Zipf($\alpha = 1.1$)	10	10	10	5	5	5	5	5	5	5
Zipf($\alpha = 1.2$)	10	10	10	5	5	5	15	20	5	5
Zipf($\alpha = 0.765$)	5	5	5	5	5	5	5	5	10	5
Zipf-M($\alpha = 1.42, \nu = 23$)	5	5	5	5	5	15	5	25	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	5	5	5	5	5	5	5	5	5	5

Table 5.20: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios

Bubble-Insert, $M = 11000$ Converged Results	Cache Size									
	11	22	44	88	176	352	704	1408	2816	5632
Zipf-M($\alpha = 1.0, \nu = 5$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.4, \nu = 69$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.6, \nu = 137$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.8, \nu = 227$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 2.0, \nu = 336$)	5	5	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.8$)	5	5	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.9$)	5	5	5	5	5	5	5	5	5	5
Zipf($\alpha = 1$)	10	10	10	10	10	5	10	5	5	5
Zipf($\alpha = 1.1$)	10	10	10	10	5	5	5	5	5	5
Zipf($\alpha = 1.2$)	10	10	10	5	5	10	5	10	5	5
Zipf($\alpha = 0.765$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.42, \nu = 23$)	5	5	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	5	5	5	5	5	5	5	5	5	5

Table 5.21: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios

5.7.3.1.2 The small dataset where $M = 2500$ gives a perspective different from the large dataset due to the relative rotational properties relative to M as mentioned previously. Tables 5.22 and 5.23 demonstrate the optimal insertion indexes for Bubble-LRU and Bubble-Insert. It is interesting to observe the difference in optimal insertion index when the popularity distribution is Zipf $\alpha = 1$ and larger. Through smaller cache

Bubble-LRU, $M = 2500$ Converged Results	Cache Size							
	10	20	40	80	160	320	640	1280
Zipf-M($\alpha = 1.0, \nu = 5$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.4, \nu = 69$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.6, \nu = 137$)	5	5	5	5	5	5	5	10
Zipf-M($\alpha = 1.8, \nu = 227$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 2.0, \nu = 336$)	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.8$)	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.9$)	5	10	10	10	5	5	5	5
Zipf($\alpha = 1.0$)	5	10	10	5	5	5	5	5
Zipf($\alpha = 1.1$)	5	10	10	10	5	5	5	5
Zipf($\alpha = 1.2$)	5	10	10	5	5	5	5	5
Zipf($\alpha = 0.765$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.42, \nu = 23$)	5	10	5	5	5	5	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	5	5	5	5	5	5	5	5

Table 5.22: Insertion Points of Bubble-LRU with the highest Cache-Hit Ratios

Bubble-Insert, $M = 2500$ Converged Results	Cache Size							
	10	20	40	80	160	320	640	1280
Zipf-M($\alpha = 1.0, \nu = 5$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.2, \nu = 26$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.4, \nu = 69$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 1.6, \nu = 137$)	5	5	5	5	5	5	5	10
Zipf-M($\alpha = 1.8, \nu = 227$)	5	5	5	5	5	5	5	5
Zipf-M($\alpha = 2.0, \nu = 336$)	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.8$)	5	5	5	5	5	5	5	5
Zipf($\alpha = 0.9$)	5	10	5	5	5	5	5	5
Zipf($\alpha = 1$)	5	10	5	10	5	10	10	10
Zipf($\alpha = 1.1$)	5	10	10	10	5	10	15	5
Zipf($\alpha = 1.2$)	5	10	10	10	10	10	10	10
Zipf($\alpha = 0.765$)	5	5	5	5	5	5	5	10
Zipf-M($\alpha = 1.42, \nu = 23$)	5	10	5	5	5	5	5	5
Zipf-M($\alpha = 1.20, \nu = 111$)	5	5	5	5	5	5	5	5

Table 5.23: Insertion Points of Bubble-Insert with the highest Cache-Hit Ratios

sizes the optimal insertion index of 10 remains, even though the larger set of items is more than four times the total number of items. This similarity appears to hold less true as the cache sizes grow, implying that relative to the rotational influence, it has a different effect on cache performance depending on the cache size and the total number of items available for request (M).

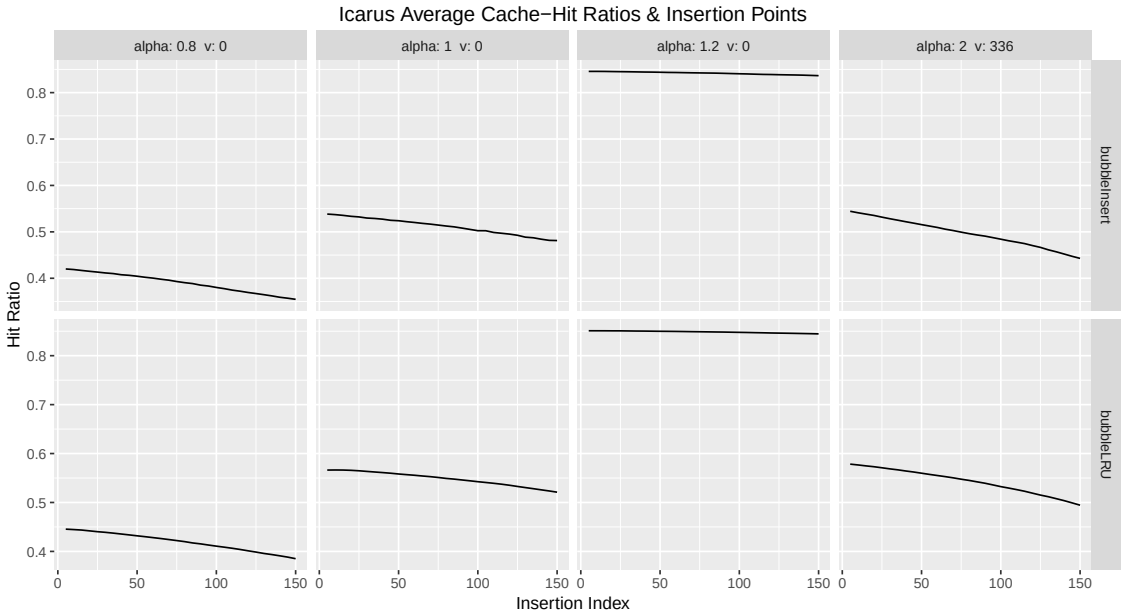


Figure 5.18: Insertion Points - Complex Request Single Cache Analysis - Cache Size: 352 — M = 11000

To illustrate the impact of varying insertion points a smaller medium cache size was chosen to be plotted in Figures 5.18 and 5.19 for the Complex Request Single Cache Analysis. When the cache sizes are large the lines are almost flat due to the large quantity of items that will permanently reside in the cache and the items affected by the varying insertion index will be those that are of a low request frequency. The Figures show an interesting pattern that suggests that higher insertion points are the most desired, however this may change if the rotation and decay of the items was less extreme. From the Figures, a definitive impact that results from the insertion index in a cache can be derived, which implies one would be required to exercise caution when choosing an appropriate insertion index as to ensure they gain the cache-hit ratio desired.

5.7.3.2 Complex Request Single Cache Analysis Cache-Hit Ratio Results

As for the results of the Complex Request Single Cache Analysis we introduce Bubble besides all other cache eviction algorithms used which are LFU, LRU, RAND and FIFO. Bubble-Insert and Bubble-LRU will only be considered when an insertion index

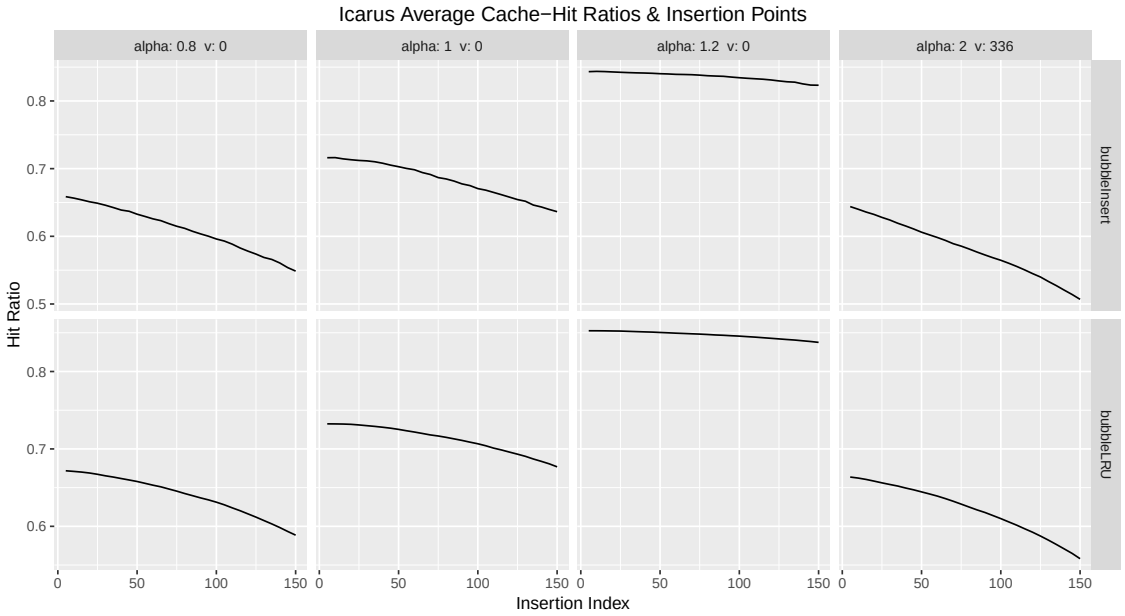


Figure 5.19: Insertion Points - Complex Request Single Cache Analysis - Cache Size: 2500 — M = 2500

is possible which is when the total cache size c is greater than the lowest insertion index tested (5).

5.7.3.2.1 Pseudo-Real Video Popularity Distributions, as decided by results found in Chapter 3, were used to aid in measuring the effectiveness of the Bubble, Bubble-LRU and Bubble-Insert cache eviction algorithms, as well as a number of known cache eviction algorithms commonly used such as *LRU* and *FIFO*. The Complex Request Single Cache Analysis, as documented in Chapter 4, was used to measure the effectiveness of each cache eviction algorithm. The pseudo-realistic distributions considered are listen below. All other simulation details were as described previously in this Section.

1. Zipf($\alpha = 0.8$) [8]
2. Zipf($\alpha = 1$) [17, 25]
3. Zipf($\alpha = 0.9$) as found in Chapter 3
4. Zipf($\alpha = 0.765$) (motivation stated in Chapter 3)
5. Zipf-Mandelbrot($\alpha = 1.42, \nu = 23$) (motivation stated in Chapter 3)

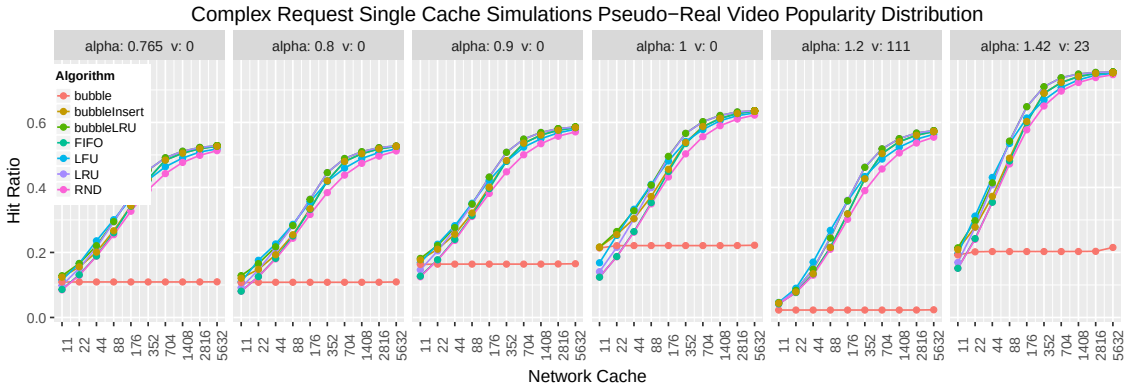


Figure 5.20: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Pseudo-Realistic Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 11000$

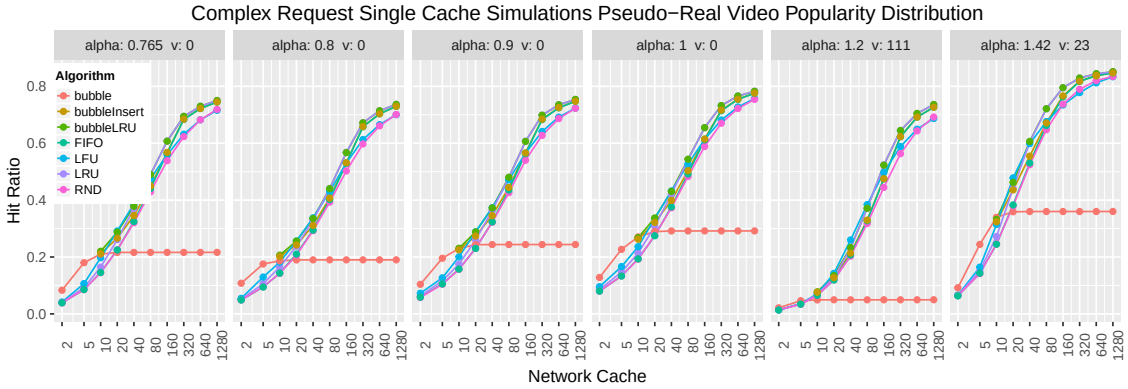


Figure 5.21: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Pseudo-Realistic Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 2500$

6. Zipf-Mandelbrot($\alpha = 1.20, \nu = 111$) (motivation stated in Chapter 3)

The Bubble cache eviction algorithm, as shown in Figures 5.20 and 5.21, exhibits some unusual, and counter intuitive behaviour when compared to other algorithms. The Bubble algorithm, even compared to the RAND algorithm, performs very poorly when the cache sizes are relatively large. The reason for this poor performance is due to fact that items in the Complex Request Single Cache simulations are available for request for a finite amount of time. New items that are introduced into the cache are required to receive two consecutive requests to be submitted to the cache for more than a single cycle. This requirement for items to maintain persistence in the cache is the barrier that stops the items that have decayed in popularity from being removed from the cache

and stops new items being introduced into the cache, unless they are items that have a large enough probability of request to be requested two times consecutively.

It is notable that, though Bubble performs poorly when the cache size is large, Bubble performs superbly when the cache size remains small. The mean difference between Bubble and the other known algorithms when the cache size is of size 10 and smaller, as shown in Table 5.24, averages a performance gain of ≈ 0.05, with a peak mean difference of 0.71. This performance can be attributed to the quick rejection of items which are requested a single time due to the items being removed as soon as a new item not yet in the cache is requested / submitted. Any items popular enough to be requested two times consecutively will remain in the cache for more than a single cycle of request, thus giving those items a greater chance of remaining in the cache. Equally important is the removal of items when they decay in popularity. Unlike LFU, which retains items until the count of requests decays to a small enough number to be overtaken by a new item, Bubble will see items bumped down in index as soon as the item below in index receives a request. This means items are likely to move frequently in a Bubble cache; implying that recency of request is more important than the measurement of historic quantity of requests, as holds importance in LFU, when items decay in popularity and when new items are frequently added and removed.

cache size	mean difference	LFU	RAND	FIFO	LRU
2	0.036	0.032	0.039	0.039	0.036
5	0.071	0.056	0.079	0.079	0.070
10	0.041	0.014	0.056	0.056	0.038
20	-0.038	-0.074	-0.015	-0.016	-0.048
40	-0.142	-0.173	-0.114	-0.118	-0.162
80	-0.253	-0.269	-0.224	-0.235	-0.283
160	-0.363	-0.358	-0.334	-0.359	-0.401
320	-0.452	-0.430	-0.420	-0.471	-0.487
640	-0.499	-0.480	-0.477	-0.514	-0.524
1280	-0.526	-0.511	-0.512	-0.537	-0.543

Table 5.24: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms in the scenario of a pseudo-realistic Popularity Distribution (Unique Item Count: 2500)

The Bubble-LRU and Bubble-Insert cache eviction algorithm performance measure-

ments are listed in Tables 5.25 and 5.26. The results for each of these algorithms are comparable and does no include any results for cache sizes equal to and lower than five. This is because the lowest insertion index used in these experiments is five.

In contrast to Bubble, Bubble-LRU holds cache-hit ratios greater or equal to the known algorithm tested in almost all situations. The average performance gained in terms of cache hit ration, as can be seen in Table 5.25, is ≈ 0.035. The performance of the Bubble-LRU can be largely attributed to incorporating the performance experienced by Bubble when cache sizes are small into a LRU cache which, when frequent change of items and their popularity occurs, adjusts well. For this reason, the difference between LRU and Bubble-LRU remain small when cache sizes are large due to the insertion index frequently being low and therefore being similar to LRU. When the cache sizes are relatively small the Bubble section of the cache, which is frequently 5 or 10 (See Section 5.7.3.1), has a larger influence on the total performance thus resulting in a greater discrepancy between the cache-hit ratio of LRU and Bubble-LRU.

Bubble-Insert appears to perform comparably to Bubble-LRU when cache sizes are small. The slightly lower cache-hit ratio experienced in Bubble-Insert when compared to Bubble-Insert when cache sizes are low is likely due to the retention that Bubble demonstrates relatively to LRU. LRU will move items that receive a request back to the top of the LRU, which is also experienced in the LRU section of the Bubble-LRU cache. This quick method and rotation of items in the LRU section will result in items being discarded from the cache much more frequently than would be true for a Bubble-Insert cache as the Bubble algorithm will see items move down only when the item in the index below a given item is requested, or when an item outside of the cache is requested if inspecting the cache below the insertion index.

This Section dove into the performance of Bubble, Bubble-LRU and Bubble-Insert and what may be the reason as to why the cache-hit ratios measured were as such, comparing the results to known cache eviction algorithms. The results used were specifically those of when the request frequency data was relatively similar to real user behaviour; referred to as “Pseudo-realistic” request distributions. The results of all of the Pseudo-

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.049	0.022	0.065	0.064	0.046
20	0.031	-0.005	0.055	0.054	0.022
40	0.026	-0.005	0.054	0.050	0.006
80	0.030	0.014	0.058	0.048	0.000
160	0.037	0.042	0.067	0.041	-0.000
320	0.034	0.056	0.067	0.015	-0.000
640	0.025	0.044	0.046	0.010	-0.000
1280	0.029	0.044	0.043	0.018	0.012

Table 5.25: Mean difference of the cache-hit ratios between BubbleLRU and other Cache Eviction Algorithms in the scenario of a pseudo-realistic Popularity Distribution (Unique Item Count: 2500)

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.044	0.017	0.059	0.059	0.041
20	0.015	-0.021	0.038	0.038	0.005
40	-0.005	-0.036	0.022	0.019	-0.025
80	-0.010	-0.027	0.018	0.007	-0.040
160	-0.001	0.003	0.028	0.002	-0.039
320	0.020	0.042	0.052	0.001	-0.014
640	0.029	0.048	0.051	0.014	0.004
1280	0.011	0.026	0.025	0.000	-0.006

Table 5.26: Mean difference of the cache-hit ratios between BubbleInsert and other Cache Eviction Algorithms in the scenario of a pseudo-realistic Popularity Distribution (Unique Item Count: 2500)

realistic request distributions were summarised and each known algorithm was directly compared to Bubble, Bubble-LRU and Bubble-Insert. The distributions were varied, as would be expected in the real world with varying video delivery services, providing an average which gives a good indicator of the effectiveness of the algorithms tested.

The results indicate that Bubble would be a good consideration in a pseudo-realistic environment for when the cache size remains small. This can be measured to be of a size until the items at the popularity index of the size of the bubble cache, if we list the available items from most popular to least, to have a low probability of consecutive requests. Exactly how low will depend largely on the distribution and decay used. In the case of the distribution Zipf($M = 2500, \alpha = 1$) in these experiments the cache size stopped increasing the cache-hit ratio when it reached a size of ≈ 20 which is when the consecutive request probability of item 21 was ≈ 0.0000321.

The results for Bubble-Insert and Bubble-LRU appears to be similar in performance in the pseudo-realist test results with a small performance increase in the Bubble-LRU results. Bubble-LRU cache eviction algorithm should be highly considered when rotation is apparent in the environment for which it could be considered, as well as when the popularity distributions found to exist in the environment is similar to the pseudo-realistic distributions used in this Section. The performance experienced in terms of cache-hit ratio for the Bubble-LRU cache eviction algorithm exceeds all other algorithms by a large margin with few exceptions.

5.7.3.2.2 Zipf Video Popularity Distributions were testing with increasing α parameters in the Markov-Chain Analysis tool to provide insight into the effectiveness that can be expected when they are submitted to Bubble, Bubble-LRU and Bubble-Insert, as well as a number of known algorithms such as the LRU and FIFO cache eviction algorithms. The increasing α parameters in a Zipf-like distribution submitted to the test provide results that give insight into the eviction algorithm one may prefer if a range of scenarios. The Zipf parameters associated with video delivery are thought to range between $\alpha : 0.8$ [8] and $\alpha : 1$ [17, 25]. Other systems, not exclusively delivering video objects, may follow more extreme power-law distributions thus creating the potential necessity to test algorithms Bubble, Bubble-LRU and Bubble-Insert with more extreme α variables that are greater than 1 as are described in this Section.
The Zipf request distributions submitted to the Markov-Chain analysis are listed below:

1. Zipf($\alpha = 0.8$)
2. Zipf($\alpha = 0.9$)
3. Zipf($\alpha = 1$)
4. Zipf($\alpha = 1.1$)
5. Zipf($\alpha = 1.2$)

The Bubble eviction algorithm appears to exhibit the same characteristics as those that have previously been mentioned in the discussion in the previous Section regarding performance of Bubble in a Pseudo-realistic environment. Bubble appears to perform extremely well, relative to other algorithms tested, when the cache size remains low due

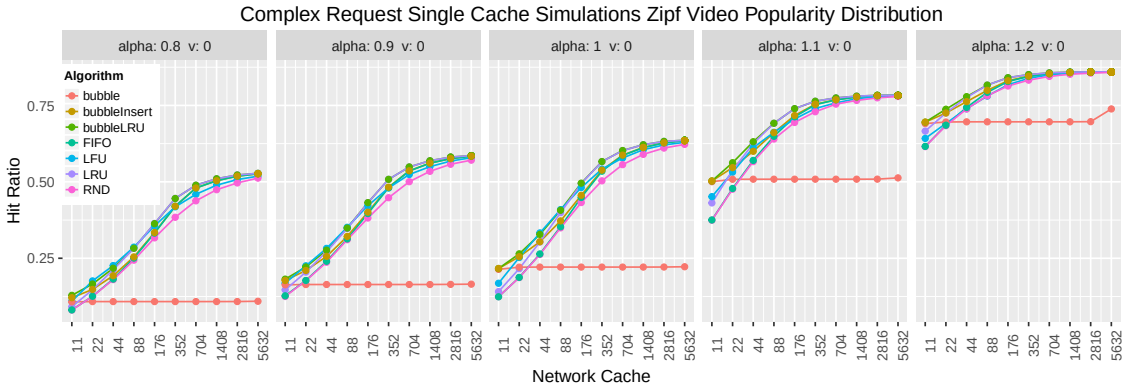


Figure 5.22: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 11000$

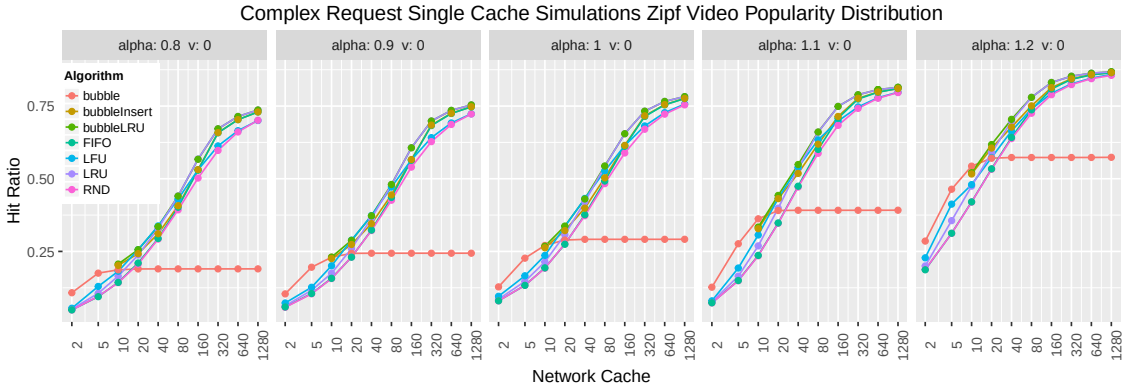


Figure 5.23: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 2500$

to the quick rejection of items that are no longer requested and the fact that items that receive a single request will not remain in the cache for more than a single request cycle.

Bubble-LRU appears to perform best in all situations tested. It is interesting to note that, as the Zipf popularity distribution becomes steeper with the increase of the α variable, the difference between the algorithms in terms of performs appears to become smaller. This appears to be true for both sets tested, where $M = 11000$ or $M = 2500$, as can be seen in Figures 5.22 and 5.23. This is to be expected with fewer items receiving an increasing amount of the total requests, thus making it easier to assess for an algorithm which items are to be contained in the cache to increase the cache-hit ratio as a result. The reason as to why Bubble-LRU produces as greater cache-hit ratio when compared

cache size	mean difference	LFU	RAND	FIFO	LRU
2	0.055	0.044	0.061	0.061	0.055
5	0.092	0.062	0.109	0.109	0.090
10	0.069	0.038	0.089	0.088	0.059
20	-0.007	-0.041	0.018	0.017	-0.022
40	-0.107	-0.131	-0.081	-0.084	-0.132
80	-0.211	-0.222	-0.185	-0.196	-0.243
160	-0.309	-0.302	-0.283	-0.306	-0.344
320	-0.381	-0.364	-0.354	-0.396	-0.411
640	-0.418	-0.404	-0.400	-0.429	-0.439
1280	-0.439	-0.429	-0.428	-0.447	-0.453

Table 5.27: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 2500)

to Bubble is discussed in the Pseudo-realistic Section.

Bubble-Insert appears to exhibit a very similar behavioural pattern as Bubble-LRU as can be seen in Figures 5.22 and 5.23 with a slightly lower average cache-hit ratio. This can be reasoned to be because of the nature of LRU which is able to eject items more rapidly than Bubble as further discussed in the Pseudo-Realistic Section of the Complex Request Single Cache result discussion.

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.062	0.031	0.083	0.082	0.052
20	0.045	0.011	0.070	0.069	0.030
40	0.033	0.009	0.059	0.056	0.009
80	0.032	0.021	0.058	0.048	0.000
160	0.035	0.041	0.061	0.037	-0.000
320	0.030	0.047	0.057	0.015	-0.000
640	0.029	0.044	0.047	0.018	0.008
1280	0.028	0.038	0.039	0.020	0.014

Table 5.28: Mean difference of the cache-hit ratios between BubbleLRU and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 2500)

5.7.3.2.3 Zipf-Mandelbrot Video Popularity Distributions modelled to closely approximate the Zipf-like distribution $\alpha : 0.8$ were submitted to the Markov-Chain Analysis method to give insight into the performance one can expect from eviction algorithms Bubble, Bubble-LRU and Bubble-Insert, as well as more commonly know cache eviction

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.057	0.026	0.078	0.077	0.047
20	0.032	-0.002	0.056	0.056	0.016
40	0.005	-0.019	0.031	0.028	-0.019
80	-0.005	-0.016	0.022	0.011	-0.036
160	0.001	0.008	0.027	0.004	-0.034
320	0.017	0.034	0.044	0.002	-0.013
640	0.012	0.026	0.030	0.001	-0.009
1280	0.008	0.018	0.020	0.000	-0.005

Table 5.29: Mean difference of the cache-hit ratios between BubbleInsert and other Cache Eviction Algorithms in the scenario of a Zipf Popularity Distribution (Unique Item Count: 2500)

algorithms LRU and FIFO, in scenarios with ranging Zipf-Mandelbrot object request distributions. The Zipf-Mandelbrot distributions were modelled to closely approximate the Zipf-like distribution with parameter $\alpha : 0.8$. α parameter 0.8 was chosen on the basis that it is between the Zipf distribution α variables found to most closely approximate the TV catch-up and VoD empirical request distributions as demonstrated in Sections 3.4.2.2 and 3.4.2.1 which uses the KL and PCS comparison methods.

The Zipf-Mandelbrot probability distribution parameters submitted to the Markov-Chain analysis tests to inspect the effectiveness of the aforementioned cache eviction algorithms are listed as follows:

1. Zipf-Mandelbrot($\alpha = 1.0, \nu = 5$)
2. Zipf-Mandelbrot($\alpha = 1.2, \nu = 26$)
3. Zipf-Mandelbrot($\alpha = 1.4, \nu = 69$)
4. Zipf-Mandelbrot($\alpha = 1.6, \nu = 137$)
5. Zipf-Mandelbrot($\alpha = 1.8, \nu = 227$)
6. Zipf-Mandelbrot($\alpha = 2.0, \nu = 336$)

The Bubble algorithm, as shown in Figures 5.24 and 5.25 appears to largely demonstrate the same strengths and weaknesses as in the Pseudo-Realistic environments as the Zipf environments discussed here. It appears Bubble suffers from inability to discard old items due to its inability to acquire newly introduced items in a large cache unless they exceed a probability in request that is likely to see them experience receiving two consecutive requests, as is discussed in more detail in the Pseudo-Realistic results

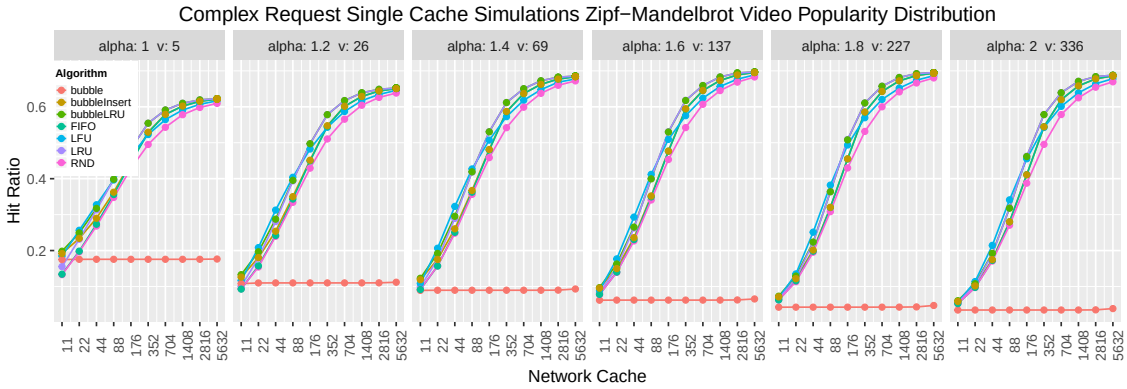


Figure 5.24: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf-Mandelbrot Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 11000$

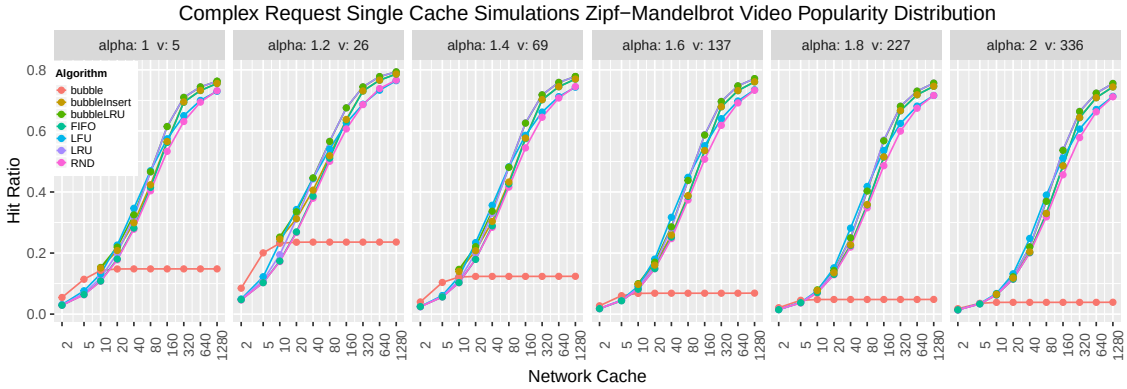


Figure 5.25: Cache-Hit Ration for each Cache Eviction Algorithm. Each Figure Displays the results for a different Zipf-Mandelbrot Video Request Distribution in a Complex Request Single Cache Analysis environment — $M = 2500$

Section. The cache-hit ratio of Bubble compared to other algorithms can be seen in more detail in Table 5.30.

Bubble-LRU appears to benefit greatly from the presence of the Zipf-Mandelbrot popularity distribution. As the difference between Bubble-LRU and the other observed known algorithms appears to remain large, even with changing α and ν values of the distributions. This is made more clear in Figure 5.25 where a smaller total set of items is used. There is a visible separation between Bubble-LRU, Bubble-Insert and the other algorithms. This can be seen more clearly in Tables 5.31 and 5.32. The discrepancy between Bubble-LRU and Bubble-Insert is small, however it is easy to see that Bubble-LRU is the preferred algorithm if applied with the correct insertion index

cache size	mean difference	LFU	RAND	FIFO	LRU
2	0.016	0.016	0.016	0.016	0.016
5	0.035	0.030	0.037	0.037	0.034
10	0.002	-0.012	0.009	0.008	0.001
20	-0.075	-0.101	-0.059	-0.060	-0.078
40	-0.185	-0.222	-0.158	-0.162	-0.197
80	-0.317	-0.348	-0.283	-0.294	-0.344
160	-0.450	-0.454	-0.412	-0.441	-0.492
320	-0.555	-0.535	-0.516	-0.575	-0.592
640	-0.608	-0.589	-0.585	-0.623	-0.637
1280	-0.639	-0.623	-0.624	-0.650	-0.659

Table 5.30: Mean difference of the cache-hit ratios between Bubble and other Cache Eviction Algorithms in the scenario of a Zipf-Mandelbrot Popularity Distribution (Unique Item Count: 2500)

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.026	0.012	0.033	0.033	0.025
20	0.017	-0.009	0.032	0.031	0.013
40	0.015	-0.022	0.042	0.038	0.003
80	0.026	-0.004	0.060	0.050	-0.001
160	0.041	0.037	0.079	0.050	-0.001
320	0.037	0.057	0.076	0.016	-0.000
640	0.028	0.048	0.052	0.013	-0.000
1280	0.020	0.036	0.036	0.009	0.000

Table 5.31: Mean deviation between BubbleLRU and other Cache Eviction Algorithms in the scenario of a Zipf-Mandelbrot Popularity Distribution (Unique Item Count: 2500)

if the popularity distribution in the environment is Zipf-Mandelbrot with resemblance to a Zipf distribution where $\alpha = 1$ as not a single mean difference measured in the Zipf-Mandelbrot simulations saw Bubble-LRU receive a lower cache-hit ratio than any other algorithm tested. The reason as to why this is, is likely the same as was explained in the test results from the Pseudo-realistic results.

5.7.3.3 Complex Request Single Cache Simulation Conclusion

Bubble, Bubble-LRU and Bubble-Insert are, in this Chapter, compared against algorithms of similar or greater operational costs in a video on demand setting. A video on demand system may show a variety of characteristics, thus presenting the need to test a number of them to identify how the presented algorithms may perform under difference

cache size	mean difference	LFU	RAND	FIFO	LRU
10	0.022	0.009	0.030	0.029	0.022
20	0.005	-0.021	0.021	0.020	0.002
40	-0.012	-0.049	0.015	0.010	-0.024
80	-0.019	-0.050	0.015	0.004	-0.046
160	-0.007	-0.011	0.030	0.001	-0.049
320	0.021	0.041	0.060	0.001	-0.016
640	0.015	0.035	0.039	0.000	-0.013
1280	0.011	0.027	0.026	0.000	-0.009

Table 5.32: Mean difference of the cache-hit ratios between BubbleInsert and other Cache Eviction Algorithms in the scenario of a Zipf-Mandelbrot Popularity Distribution (Unique Item Count: 2500)

circumstances. This experiment had, the functionality to add, remove items, as well as have items decay in their probability of request throughout the experiment. When summarising the results from the Zipf, Zipf-Mandelbrot and Pseudo realistic simulations it can be concluded that Bubble-LRU is the most effective algorithm in all cases for which it was tested. This is so due to the adaptive nature of Bubble and LRU combined.

Bubble was tested for a number of scenarios for which Bubble-LRU and Bubble-Insert were not due to the restriction of insertion index. Bubble was the most successful algorithm, producing the greatest cache-hit ratio, across all distributions it was tested when the cache size remained small. Scenarios where cache storage is expensive and cache sizes are required to remain of a limited size, Bubble would provide the most appropriate algorithm thus concluding Bubble and Bubble-LRU to be the most appropriate algorithms if the scenario in which a caching algorithm is required experiences a changing pool of items and changing popularity of those items whilst demonstrating a Zipf or Zipf-Mandelbrot popularity distribution.

5.7.4 Exploration of Insertion Index

Bubble-LRU and Bubble-Insert have the unique property of an insertion index. To explore the impact of the location of the insertion index a separate set of experiments are introduced in this section that leverage the Complex Request Single Cache Simulation introduced in this Chapter. The experiment assesses the cache-hit ratio of each index of the cache instead of the entire cache to more closely inspect the impact of the insertion

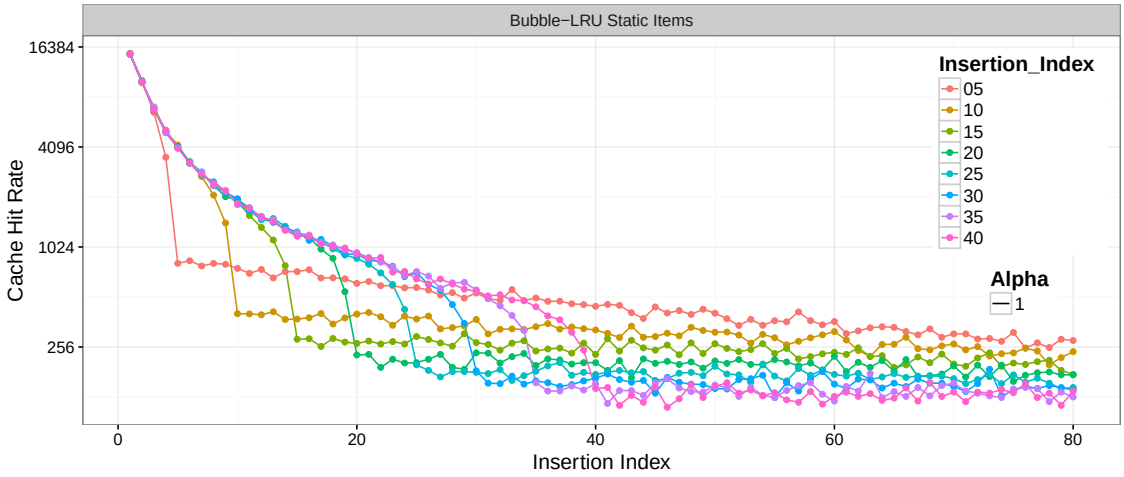


Figure 5.26

index.

An assumption expressed in previous sections was that the Bubble section, in Bubble-LRU and Bubble-Insert cache, provides a section of cache that provides superior retention of the more popular items in the dataset. This assumption can be confirmed when inspecting Figure 5.26 and Figure 5.27 which holds cache hit rates of each index of the cache of a simulation of a static set of items following a Zipf distribution for Bubble-Insert and Bubble-LRU for a variety of insertion indexes of a cache of size 80. Additionally these figures confirm that the cache-hit ratio of the Bubble section of the Bubble-Insert cache ahead of the insertion index, where index is greater than the insertion index, does indeed receive a reduced cache-hit ratio when compared to the LRU section of the Bubble-LRU cache.

Just as Figures 5.26 and 5.27 demonstrate, Figures 5.28 and 5.29 show a very comparable result however with a less clear benefit of the variability of insertion index.

Figures 5.26, 5.27, 5.28 and 5.29 all show the cache-hit rate of each index within a cache of cache size 80 with a variety of insertion indexes. The results appear to follow a pattern where the Insertion Index of the items shifts the area of the cache where the highest cache hit rates are measured. It is worth noting that the highest summed hit rates for all Insertion Indexes varies between the Static pool of requestable items and the rotating pool of requestable items simulation results. In the case of the rotating

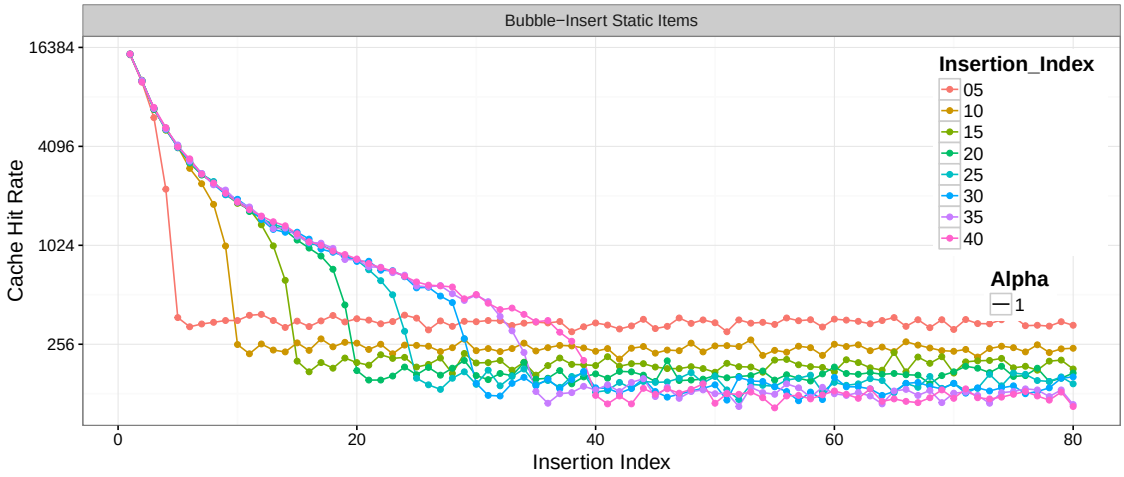


Figure 5.27

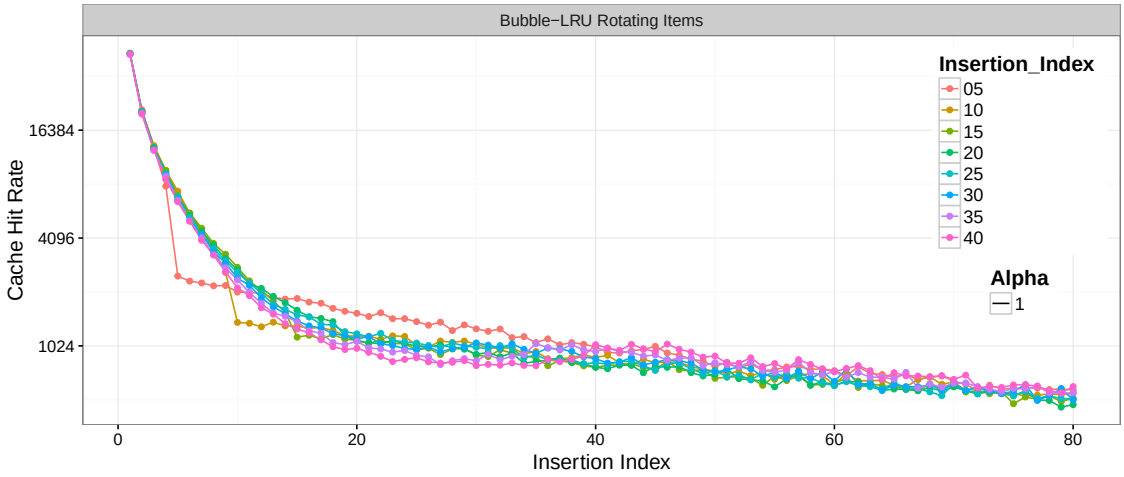


Figure 5.28

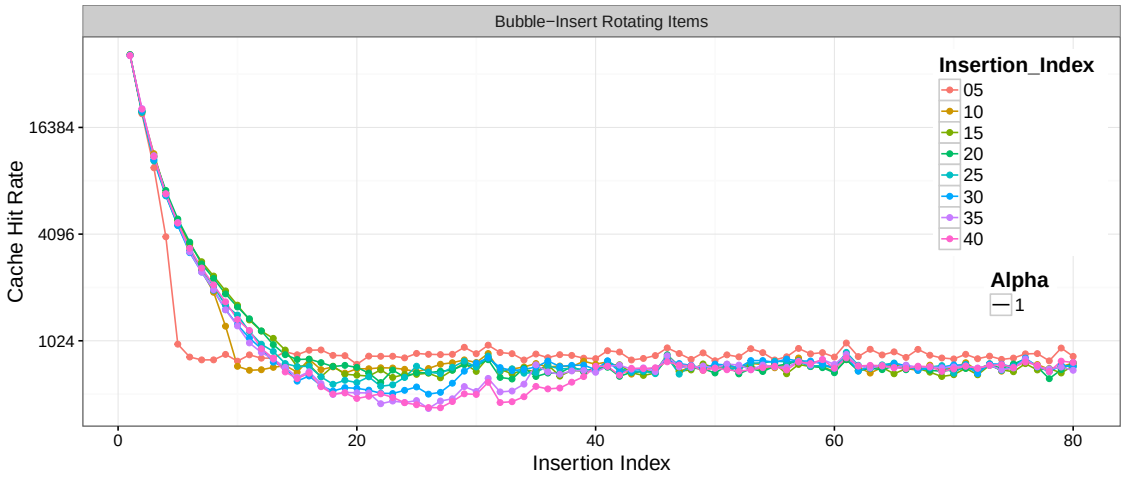


Figure 5.29

item pool results (marked as 'story') the highest hit-rate is experienced approximately at Insertion Index 15. In the case of the results of the static pool of items the insertion index is found to be greater as the insertion index is of a larger index, in this case where the Insertion Index is equal to 40. No concrete pattern of behaviour appeared to be derivable from the results.

5.8 Conclusion Bubble Eviction Algorithm

This chapter introduced the Bubble cache eviction algorithm. Additionally to the introduction of Bubble, two variations of Bubble were also introduced as Bubble-LRU and Bubble-Insert. The Bubble algorithm takes inspiration from the Bubble sort algorithm which iteratively compares items and swaps them based on the predefined rule. Bubble also swaps items however, it interprets them based on a recency of requests experienced, thus providing the requested item with a greater value, swapping it with the item preceding it. This mechanism is introduced in this Chapter as a simple cache eviction algorithm with potential to rival existing, better known algorithms such as LRU, FIFO, RAND and even LFU. Bubble-LRU and Bubble-Insert are variations in which the insertion of new items, which is at the lowest index in Bubble, is changed to be at alternative indexes with a change to the proceedings below the alternative index to follow the LRU method in the Bubble-LRU algorithm.

Bubble, Bubble-LRU and Bubble-Insert are compared to a number of algorithms in three different methods of testing. They are:

1. Markov Chain Analysis
    (a) Analytical
    (b) Single Cache
    (c) No decay of probability of objects
    (d) No removal/introduction of available objects
2. Information Centric Networking Simulation (Icarus)
    (a) Simulation
    (b) Topology

(c) Placement Policies

(d) No decay of probability of objects

(e) No Removal / Introduction of available objects

3. Complex Request Single Cache Simulation

(a) Simulation

(b) Single Cache

(c) Decay of items

(d) Removal.Introduction of items

This slightly simplified list incorporates a number of the most influential factors that make up the methods used for testing the cache eviction algorithms introduced. For this reason, comparing sets of results is to be approached with caution.

The Analytical method of testing, due to the extremely expensive method of testing, cannot encompass large sets of available items as well as a large cache size. For this reason the results are drawn from a set of results in which the cache size is small and the set of available items is relatively small. An additional challenge is the inability for Bubble to converge when a cache of a large size is used. The Icarus simulation environment contains complexity in the method of delivery of items in a network. The simulator is constructed in such a way that considers topology, routing strategies, cache eviction algorithms and popularity distribution that does not change through the simulation. The Complex Request Single Cache Simulation does not include routing strategies and network topology, but instead holds a single cache through which the objects are moved when requested.The pool of available objects changes with a decay introduced to the objects in the simulation. This changing pool of items requires cache eviction algorithms to, as well as aim to contain the most frequently requested items, also dispel those that decay and are removed from a Video on Demand environment. This makes the Complex Request Single Cache Simulation unique.

In a small test environment, such as the one of the Analytical Markov Chain tests performed, Bubble, Bubble-LRU and Bubble-Insert are experienced to provide a superior cache eviction algorithm over the other algorithms tested. The limiting analysis

possible with Bubble does mean that we cannot conclude Bubble to provide us with a cache-hit ratio towards which it trends for the larger cache sizes. Bubble-LRU and Bubble-Insert do not suffer from this inability to converge for larger cache sizes, thus making it possible to declare that, within the scope of the possible Markov-Chain analysis performed in this Chapter, Bubble and its variants provide effective alternative algorithms that should be encouraged to be tested and used in Video in Demand Environments.

In the Icarus environment we experience Bubble performing better than all other algorithms tested in a limited number of situations. It was found that Bubble-LRU and Bubble-Insert rarely surpass the cache-hit ratio of LFU which was most often measured to be the most effective algorithm tested in the Icarus environment. This is likely due to the stationary set of objects available for request. It is interesting to note that for when cache sizes ranged between 0.016 and 0.128 (40% of the tests), Bubble still performed better than LFU, thus suggesting Bubble to be a contender to be the most appropriate algorithm in a large amount of situations that resemble the Icarus simulation environment.

The Complex Request Single Cache Simulation environment was used to test Bubble and its variants; Bubble-LRU and Bubble-Insert. The results are drastically different from the Icarus results and the results found in the Markov-Chain analysis, likely due to the changing nature of the items available for request. The simulation results suggest that Bubble is appropriate to apply when cache sizes remain small. The cache-hit ratio experienced by Bubble exceeded other algorithms by a large margin, sometimes even as much as 150% of the next best producing cache-hit ratio algorithm. When the cache size is increased, the Bubble algorithm interestingly hits a maximum cache-hit ratio which, no matter the size of the cache, it seems unable to increase beyond. Bubble-LRU and Bubble-Insert were found to remain relevant beyond the small cache-size with a cache-hit ratio that exceeded all other algorithms in most tests performed. Bubble-LRU performed very similarly to Bubble-Insert in terms of cache-hit ratio with an improvement of cache-hit ratio.

As to conclude, Bubble would be the most appropriate to consider in an environment such as that of the Complex Request Single Cache Simulation environment when the cache size remains small. Bubble-LRU provides a better solution than the other known algorithms testing when the cache size is expanded with the caveat that where implemented, the appropriate insertion index would need to be applied which can have an adverse effect if not chosen properly.

6

Conclusion

This Thesis set to achieve a greater understanding of video request data and its intricacies, as well as to propose methods to alleviate some of the strain VoD currently has on the Internet. Two data-sets were analysed to provide a closer look at the structures observable in VoD data. The research community considers two separate models to resemble VoD request distributions closely. This Thesis identifies a single superior model for replication of VoD request distributions based on the provided data. Additionally, a request generator was created introducing factors such as decay of popularity and removal and introduction of objects, which previously were commonly overlooked influencers in VoD data. The Request Generator, as well as an ICN simulation environment and a Markov Chain Analysis method were used to simulate VoD traffic to assess three novel cache eviction algorithms, namely Bubble, Bubble-Insert and Bubble-LRU, in a VoD environment in an effort to provide a platform for improved VoD delivery.

6.1 Motivation for work

As previously mentioned in the introduction of this Thesis, the motivation comes from wanting to understand video request data and suggesting an area of improvement that may impact video streaming directly. BT provided popularity request data in the form of a two popularity distributions. The scope of the work was partially focused around attributing to the discussion surrounding the debate of which model, Zipf or Zipf-Mandelbrot is most appropriate for modelling video popularity distribution. In the effort to conclude which model provides the most appropriate model to replicate video request distributions a number of methods had to be found that demonstrate, in a statistically significant manner, techniques for comparing a model to another data-set.

A goal also set was to further the communal understanding of the characteristics surrounding video requests in an effort to recreate them to the best of our abilities, as well as create a novel cache eviction algorithm to be implementable in a cache-enabled network infrastructure focused purely on reducing traffic associated with VoD.

6.2 Summary

Chapters 3, 4 and 5 contain the primary research pieces. Chapter 5 contains novel cache eviction algorithms with a primary focus in helping grant a reduction of VoD traffic in a cache enabled network such as a network that has implemented the ICN or CCN network protocols. To achieve a network model accurate enough to conclude the effectiveness of the novel cache eviction algorithms, a great understand had to be had of video request generation in real VoD systems. Deficiencies exist within the research community with regards to VoD data due to data required to draw key characteristic attributes of VoD request data being sparse and valuable. Although a section of the video request data can be fabricated using a number of resources and intuition, as further explained in Chapter 4, two VoD popularity distribution were provided and analysed to further the general understanding of at least this specific key characteristic within video

request data, which aided in the completion of Chapter 4 and provided a platform for replicating real request data. Chapter 5 had a primary focus on video caching. Chapter 4 helped aid progress in Chapter 5 by creating a method of replicating video request data which could be used to test cache eviction algorithms aimed at caching specifically video data.

6.3 Conclusions

As to conclude the main Chapters, this Thesis sought to gain a greater understanding of video request data and a method by which a cache enabled network could optimise its delivery in avoiding replicated video data on its network. Chapter 3 contained analysis of two VoD popularity distribution data-sets provided by BT. The wider research community partially claimed that VoD popularity distributions follow a Zipf Distribution [7–18]. Other research findings have contradicted this claim or expressed caution as to assuming the distribution follows Zipf [12, 16, 17], thus providing a need to settle this dispute. A number of methods were used in an effort to deduce which model, Zipf or Zipf-Mandelbrot, is most appropriate in an effort to replicate VoD popularity distributions. The means of matching distributions included analytical methods leveraging Kullback-Leibler divergence, Pearson Chi-Sqaured, Pearson Correlation and an ICN network simulation environment used to assess the behavioural similarities of the request behaviours of the most prominent Zipf and Zipf-Mandelbrot distribution found to match the real popularity distributions most closely. The results of all tests performed concluded that Zipf-Mandelbrot is the most appropriate model to use when replicating VoD popularity distributions with emphasis on the performance differential experienced in the Icarus ICN environment when comparing Zipf and Zipf-Mandelbrot against the real popularity distributions. Zipf-Mandelbrot resembled the real data to a much greater degree as can be quantified and seen in Table 6.1 where a mean average cache-hit ratio for all cache sizes is summarised to demonstrate the close resemblance to the real data-set Zipf-Mandelbrot shows in relation to the real data as opposed to the

	Empirical and Model — Mean Absolute Deviation			
	VoD		TV Catch-up	
Cache Size	Zipf	Zipf-Mandelbrot	Zipf	Zipf-Mandelbrot
Total Mean	0.0304	0.0037	0.0385	0.0056

Table 6.1: Difference between Cache-Hit Ratios relative to Empirical Results in test where Zipf and Zipf-Mandelbrot models were selected based on a goodness-of-fit produced in KL — This table is a reduced replicate of Table 3.6

Zipf Distribution. In determining these results the methods by which it was possible to compare models and data-sets was the largest contribution. The Pearson Chi-Squared method used, though unconventional, is a novel method of comparing two popularity distributions that would provide an appropriate additional method of comparing two models beside the more conventional method of using the Kullback-Leibler divergence.

Chapter 4 set out to create an ordered list of requests for video objects made to a CDN that reflects the changing probabilities one may expect to find in a VoD system. The key characteristics that were set out to include for each unique video object were; *Probability of Request*, *Decay of Popularity* and *Life-Time*. The process suggested to create a list of requests and requires the user to introduce the properties of items in the system at a single point of observation. Once the properties of each item are identified, the requests can be generated by a two step process. The steps are documented as being the "Storyboard Generator" (Chapter 4.3.1) and the "Request Generator" (Chapter 4.4).

The list of requests generated by the request generator here may be of use to a number of ranging application requiring a pseudo-realistic request order for items. One such example may be cache eviction policy effectiveness in a VoD setting such as for the algorithms proposed in Chapter 5. As to conclude, a successful pseudo-realistic request generator for a VoD system was designed and implemented for the purposes specified.

In Chapter 5, the Bubble cache eviction algorithm is introduced. Additionally to the introduction of Bubble, two variation of Bubble were also introduced as Bubble-LRU and Bubble-Insert. The Bubble algorithm takes inspiration from the Bubble sort

algorithm which iteratively compares items and swaps them based on the predefined rule. Bubble also swaps items however, it interprets items based on a recency of requests experienced, thus providing the requested item with a greater value swapping it with the item preceding it. This mechanism is introduced in this Thesis as a simple cache eviction algorithm with potential to rival existing, better known algorithms such as LRU, FIFO, RAND and even LFU. Bubble-LRU and Bubble-Insert are variations in which the insertion of new items, which are at the lowest index in Bubble, are changed to be at an alternative index with a change to the proceedings below the alternative index to follow the LRU method in the Bubble-LRU algorithm.

Three separate methods were used to simulate the approximate cache-hit ratios one may be expected to find in a video on demand environment with caching. Each method used had a variety of results, however, from the Markov Chain and Icarus Simulations it can be concluded that Bubble is an effective eviction algorithm. Bubble, when results were obtainable, received a greater cache-hit ratio in most scenarios with the exception of LFU which closely resembled Bubble in most cases. This means that Bubble is the most effective eviction algorithm out of all algorithms used with the same operation cost of O(n). The third method of testing Bubble and it's variants saw them submitted to the Complex Single Cache Request Generator created in Chapter 4. This method did not see a static pool of items available for request but instead saw a changing pool of available items as well as the introduction of popularity decay. This new dimension saw Bubble perform better than all other cache eviction algorithms tested when the cache size was very small, however not produce positive results when the cache size was medium or large. In the scenarios where Bubble did not perform, Bubble-LRU appeared to apply the performance Bubble applied in the small caches together with the effectiveness of LRU and performed better than all other algorithms observed.

To conclude, Bubble is the most appropriate cache eviction algorithm when cache sizes remain small based on the results observed in Chapter 5. When cache sizes are

large, a cache eviction algorithm such as Bubble-LRU would be worthy of investigation as it can achieve a great cache-hit ratio in a range of situations.

6.4 Future Work

Chapter 3 has introduced a number of methods considered appropriate for testing and in this chapter two power-law distributions are introduced and compared against a real video request distribution data-set. These methods of testing have demonstrated that there is indeed a superiour model between the two models submitted. The comparison methods can extend to other models that were not in the scope of this Thesis, such as log-normal [17, 33, 34] and k-transform [14], and many others.

Bubble, Bubble-Insert and Bubble-LRU were introduced in this Thesis as suggested technologies to be applied to reduce traffic on a network over which video delivery is experienced and local caching is available. Although video delivery is responsible for a large amount of the total traffic on many network infrastructures, it is not the only subject area in which these novel algorithm can be applied. The Bubble algorithm and its variants may be found to be beneficial in multiple areas where simple caching algorithms are applied, thus potentially contributing to greater cache efficiency beyond the VoD scope in which Bubble was demonstrated to be effective. The Zipf distribution has appeared in many distributed forms of content such as Images, HTML pages and many other types of media available for request on the World Wide Web (WWW) [25–28], thus making all of these types of media attractive areas of investigation for the application of Bubble caching.

Chapter 4 set out to create an ordered list of requests for video objects made to a CDN that reflects the changing probabilities one may expect to find in a VoD system. This system, though considered pseudo real, may provide a platform on which assumptions made about video request data may be scrutinised when user observed request data is not available. Additionally to providing a method for proving, or disproving assumptions, statistics may be drown from a simulation environment in aid to model a real system - should the general characteristics, such as user count, frequency of request

per user and the popularity distribution of the items contained in the video delivery service, be known or predictable.

Video delivery is a subject that requires more exploration as currently, effects of decay, limited lifetime of items and methods used for observation of popularity distributions, leave much to be desired. The limiting factor in effectively modelling all these factors, not all of which are listed here as they are now all known or well understood, is the availability of VoD data. If video request data were to be obtained, it would be suggested to attempt to replicate VoD requests in Chapter 4 and observe for any improvements.

6.5 Implications of Work

The novel work introduced in this Thesis consist of identifying the Zipf-Mandelbrot distribution within Video Request Distributions and the introduction of the Bubble eviction algorithms and variants of the Bubble eviction algorithms. These observations and creations are novel and are applicable to VoD systems in the methods introduced in this Thesis.

The danger of simply assuming a Zipf-like distribution instead of a Zipf-Mandelbrot distribution as suggested in this Thesis for video consumption, without supporting evidence, is that the resultant model may not be sufficiently representative of the user demands. Assuming a badly fitting model may lead to sub-optimal design choices in areas, such as service planning/provisioning, utilization of network resources, accommodation of SLAs. To enable users to correctly simulate video request data to a greater accuracy than previously possible could create for fewer predictive errors for anyone seeking to create a large scale VoD system.

The Bubble Cache Eviction algorithm and variations provide alternative cache eviction algorithms to, for example, LRU with a potential to achieve greater traffic reduction in VoD environments with caching enabled than was previously possible with known simple cache eviction algorithm. The Bubble algorithm may be found to be beneficial in multiple areas where simple caching algorithms are applied, thus potentially contribut-

ing to greater cache efficiency beyond the VoD scope in which Bubble was demonstrated to be effective.

Additional to these contributions a number of methods were used, some of which were novel, to conclude the listed results, such as the leveraging Kullback-Leibler divergence, Pearson Chi-Sqaured, Pearson Correlation and an ICN network simulation environment as well as the creation of a novel video request generation platform that provides a diverse request generation method designed to incorporate a large set of variables associated with video delivery systems that may be present in an observed video request data-set. These methods aided in determining all results above and are diversely applicable.

A

Supplementary Evaluations and Results

A.1 Markov-Chain Empirical Analysis Results - Bubble-LRU & Bubble-Insert

Zipf Model	Algorithm	Hit Ratio / Cache Size (Insertion Point) 4	5	6	7	8	9	10
	LRU	0.124	0.151	0.177	0.201	0.224	0.247	0.267
	FIFO	0.118	0.141	0.164	0.185	0.205	0.224	0.242
Zipf($\alpha = 0.8$)	Bubble	0.239	0.282	0.317	0.357	*1	*1	*1
	B-LRU	0.171	0.229	0.275	0.312	0.325	0.336	0.353
	B-Insert	0.171	0.230	0.276	0.312	0.326	0.342	0.354
	LRU	0.163	0.196	0.227	0.256	0.283	0.308	0.331
	FIFO	0.152	0.181	0.207	0.231	0.253	0.274	0.294
Zipf($\alpha = 0.9$)	Bubble	0.295	0.341	0.379	*0.415	*1	*1	*1
	B-LRU	0.219	0.284	0.334	0.373	0.386	0.402	0.426
	B-Insert	0.220	0.285	0.334	0.373	0.388	0.404	0.434
	LRU	0.212	0.252	0.289	0.321	0.351	0.378	0.402
	FIFO	0.195	0.228	0.258	0.285	0.310	0.332	0.354
Zipf($\alpha = 1.0$)	Bubble	0.355	0.404	0.443	*0.475	*1	*1	*1
	B-LRU	0.273	0.344	0.397	0.413	0.451	0.465	0.475
	B-Insert	0.275	0.345	0.397	0.438	0.452	0.466	0.482
	LRU	0.271	0.317	0.358	0.393	0.425	0.452	0.478
	FIFO	0.246	0.283	0.317	0.346	0.372	0.397	0.419
Zipf($\alpha = 1.1$)	Bubble	0.418	0.469	0.509	*0.541	*1	*1	*1
	B-LRU	0.334	0.408	0.462	0.503	0.517	0.545	0.563
	B-Insert	0.336	0.409	0.462	0.503	0.518	0.547	0.565
	LRU	0.337	0.389	0.432	0.469	0.501	0.528	0.552
	FIFO	0.303	0.344	0.380	0.411	0.439	0.464	0.486
Zipf($\alpha = 1.2$)	Bubble	0.482	0.533	0.573	*0.606	*0.776	*1	*1
	B-LRU	0.397	0.473	0.527	0.568	0.582	0.614	0.624
	B-Insert	0.400	0.474	0.528	0.568	0.584	0.614	0.625

Table A.1: The All Cache Eviction Algorithm Markov-Chain Cache-Hit Ration results for Zipf scenario results

Real Scenarios Model	Algorithm	Hit Ratio / Cache Size (Insertion Point) 4	5	6	7	8	9	10
Zipf$^{(\alpha=0.765)}$	LRU	0.112	0.138	0.162	0.185	0.207	0.228	0.248
	FIFO	0.108	0.130	0.151	0.171	0.190	0.208	0.226
	Bubble	0.221	0.262	0.297	*0.352	*1	*1	*1
	B-LRU	0.156	0.212	0.256	0.292	0.304	0.315	0.332
	B-Insert	0.156	0.212	0.256	0.292	0.305	0.317	0.334
Zipf$^{(\alpha=0.9)}$	LRU	0.163	0.196	0.227	0.256	0.283	0.308	0.331
	FIFO	0.152	0.181	0.207	0.231	0.253	0.274	0.294
	Bubble	0.295	0.341	0.379	*0.415	*1	*1	*1
	B-LRU	0.219	0.284	0.334	0.373	0.386	0.402	0.426
	B-Insert	0.220	0.285	0.334	0.373	0.388	0.404	0.434
Zipf-M$^{(\alpha=1.42,\nu=23)}$	LRU	0.0619	0.0771	0.0923	0.107	0.122	0.137	0.152
	FIFO	0.0616	0.0765	0.0913	0.106	0.120	0.134	0.149
	Bubble	0.0881	0.116	*1	*1	*1	*1	*1
	B-LRU	0.0668	0.0913	0.118	0.132	0.147	0.161	0.175
	B-Insert	0.0676	0.0916	0.119	0.133	0.149	0.164	0.181
Zipf-M$^{(\alpha=1.20,\nu=111)}$	LRU	0.0423	0.0529	0.0635	0.0741	0.0846	0.0952	0.106
	FIFO	0.0424	0.0529	0.0635	0.0741	0.0846	0.0951	0.106
	Bubble	0.0448	0.0565	*1	*1	*1	*1	*1
	B-LRU	0.0428	0.0534	0.0639	0.0745	0.0851	0.0956	0.106
	B-Insert	0.0431	0.0541	0.0651	0.0763	0.0877	0.0992	0.111

Table A.2: The All Cache Eviction Algorithm Markov-Chain Cache-Hit Ration results for pseudo-real scenario results

Real Scenarios Model	Algorithm	Hit Ratio / Cache Size (Insertion Point) 4	5	6	7	8	9	10
Zipf-M$^{(\alpha=1,\nu=5)}$	Bubble	0.1338	0.1736	1	1	1	1	1
	LRU	0.0783	0.0972	0.1159	0.1343	0.1524	0.1703	0.1878
	FIFO	0.0772	0.0952	0.1129	0.1301	0.1469	0.1633	0.1794
	B-LRU	0.0933	0.1323	0.1716	0.1872	0.2026	0.2172	0.2335
	B-insert	0.0938	0.1327	0.172	0.1885	0.205	0.2222	0.2382
Zipf-M$^{(\alpha=1.2,\nu=26)}$	Bubble	0.0707	0.0927	1	1	1	1	1
	LRU	0.0533	0.0666	0.0798	0.0929	0.106	0.1191	0.1321
	FIFO	0.0532	0.0663	0.0793	0.0922	0.1049	0.1176	0.1302
	B-LRU	0.0565	0.0746	0.0958	0.1085	0.121	0.1444	0.1465
	B-insert	0.0571	0.0748	0.0959	0.1091	0.1226	0.1411	0.1505
Zipf-M$^{(\alpha=1.4,\nu=69)}$	Bubble	0.0527	0.0677	1	1	1	1	1
	LRU	0.0457	0.0571	0.0685	0.0799	0.0913	0.1026	0.114
	FIFO	0.0458	0.0571	0.0684	0.0797	0.091	0.1022	0.1134
	B-LRU	0.047	0.0584	0.0717	0.0839	0.0953	0.1071	0.119
	B-insert	0.0474	0.0592	0.0719	0.0843	0.0959	0.1094	0.1206
Zipf-M$^{(\alpha=1.6,\nu=137)}$	Bubble	0.0461	0.0583	1	1	1	1	1
	LRU	0.0429	0.0536	0.0644	0.0751	0.0858	0.0965	0.1072
	FIFO	0.043	0.0537	0.0644	0.075	0.0857	0.0964	0.107
	B-LRU	0.0435	0.0542	0.065	0.0757	0.0864	0.0971	0.1078
	B-insert	0.0439	0.055	0.0662	0.0776	0.0891	0.1007	0.1125
Zipf-M$^{(\alpha=1.8,\nu=227)}$	Bubble	0.0434	0.0545	0.064	1	1	1	1
	LRU	0.0417	0.0522	0.0627	0.0731	0.0835	0.094	0.1044
	FIFO	0.0418	0.0523	0.0627	0.0731	0.0835	0.0939	0.1043
	B-LRU	0.0421	0.0525	0.063	0.0734	0.0839	0.0943	0.1047
	B-insert	0.0424	0.0532	0.0642	0.0752	0.0864	0.0978	0.1092
Zipf-M$^{(\alpha=2,\nu=336)}$	Bubble	0.0421	0.0527	0.0618	1	1	1	1
	LRU	0.0412	0.0515	0.0619	0.0722	0.0825	0.0928	0.1031
	FIFO	0.0413	0.0516	0.0619	0.0722	0.0825	0.0928	0.1031
	B-LRU	0.0414	0.0517	0.0621	0.0724	0.0827	0.093	0.1033
	B-insert	0.0418	0.0524	0.0632	0.0741	0.0852	0.0964	0.1077

Table A.3: The All Cache Eviction Algorithm Markov-Chain Cache-Hit Ration results for Zipf-M scenario results

References

[1] Cisco Systems 2017. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20162021 White Paper. *Cisco*, pages 2016–2021, 2017.

[2] S. Uysal. Apparatus and method for transparent selection of an internet server based on geographic location of a user, December 18 2007. US Patent 7,310,686.

[3] Microsoft. Traffic Manager routing methods, 2017.

[4] Cloudflare. Cloudflare CDN. 2017.

[5] Greg Ireland. WHITE P APER Broadband and Pay TV Operators Adopt CDN Strategies to Manage Changes in Consumer Video Behavior. 2013.

[6] White Paper. Cisco Visual Networking Index : Usage. *Forecast*, pages 1–10, 2016.

[7] Xiaobo Zhou and Cheng Zhong Xu. Optimal video replication and placement on a cluster of video-on-demand servers. In *Proceedings of the International Conference on Parallel Processing*, volume 2002-Janua, pages 547–555, 2002.

[8] TR Nair and P Jayarekha. A rank based replacement policy for multimedia server cache using zipf-like law. *arXiv preprint arXiv:1003.4062*, 2(3):14–22, 2010.

[9] Ake Arvidsson, Manxing Du, Andreas Aurelius, and Maria Kihl. Analysis of user demand patterns and locality for YouTube traffic. *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, pages 1–9, sep 2013.

[10] Hongliang Yu, Dongdong Zheng, Ben Y. Zhao, and Weimin Zheng. Understanding user behavior in large-scale video-on-demand systems. *ACM SIGOPS Operating Systems Review*, 40(4):333, oct 2006.

[11] Miklos Mate, Rolland Vida, Andras Csaszar, and Attila Mihaly. Offloading video servers: P2P and/or caches? *2012 15th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6, oct 2012.

[12] Soam Acharya, Brian Smith, and Peter Parnes. Characterizing User Access To Videos On The World Wide Web. *Mmcn'00*, 2000.

[13] Ludmila Cherkasova and Minaxi Gupta. Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. *Network and operating systems support for digital audio and video*, 12:33, 2002.

[14] Wenting Tang, Yun Fu, Ludmila Cherkasova, and Amin Vahdat. Modeling and generating realistic streaming media server workloads. *Computer Networks*, 51(1):336–356, jan 2007.

[15] M. Vilas, X. G. Pañeda, R. García, D. Melendi, and V. G. García. User behaviour analysis of a video-on-demand service with a wide variety of subjects and lengths. *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference*, pages 330–337, 2005.

[16] Krishna P Gummadi, Richard J Dunn, Stefan Saroiu, Steven D Gribble, Henry M Levy, and John Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-SharingWorkload. 2003.

[17] Meeyoung Cha, Haewoon Kwak, and Pablo Rodriguez. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. *Proceedings of the 7th ...*, 2007.

[18] Jussara M. Almeida, Jeffrey Krueger, Derek L. Eager, and Mary K. Vernon. Analysis of educational media server workloads. *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '01*, (Nossdav):21–30, 2001.

[19] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bar, Alessandro Finamore, and Tanja Zseby. When youtube does not work-analysis of QoE-relevant

degradation in google CDN traffic. *IEEE Transactions on Network and Service Management*, 11(4):441–457, 2014.

[20] Yuejian Xie and Gabriel H Loh. PIPP : Promotion / Insertion Pseudo-Partitioning of Multi-Core Shared Caches. 37(3):174–183, 2009.

[21] Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. *Proceedings of the 34th SIGCSE technical symposium on Computer science education - SIGCSE '03*, 35(1):1, 2003.

[22] Bin Cheng, Xuezheng Liu, Z Zhang, and H Jin. A Measurement Study of a Peer-to-Peer Video-on-Demand System. In *International Peer to Peer Symposium (IPTPS)*, number 60433040, pages 1–6, 2007.

[23] Kun Lung Wu, Philip S. Yu, and Joel L. Wolf. Segmentation of multimedia streams for proxy caching. *IEEE Transactions on Multimedia*, 6(5):770–780, 2004.

[24] K a Hua and S Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. *Computer Communication Review*, 27(4):89–100, 1997.

[25] V. Almeida, A. Bestavros, M. Crovella, and A. De Oliveira. Characterizing Reference Locality in the WWW. *Parallel and Distributed Information Systems (PDIS)*, pages 92–103, 1996.

[26] L. Breslau, Pei Cao Pei Cao, Li Fan Li Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, 1, 1999.

[27] Carlos R Cunha and Mark E Crovella. Characteristics of WWW Client-based Traces. *Environment*, pages 1–18, 1995.

[28] Toshihiko Yamakami. A zipf-like distribution of popularity and hits in the mobile web pages with short life time. 2006.

[29] Qianqian Yang, Mohammad Mohammadi Amiri, and Deniz Gunduz. Audience retention rate aware coded video caching. *2017 IEEE International Conference on Communications Workshops, ICC Workshops 2017*, pages 1189–1194, 2017.

[30] Konrad Karolewicz and Andrzej Beben. Cloud-based Adaptive Video Streaming: Content storage vs. transcoding optimization methods. *Proceedings - IEEE Symposium on Computers and Communications*, (July):523–528, 2017.

[31] Dimitrios N Serpanos, George Karakostas, and Wayne Hendrix Wolf. Effective caching of Web objects using Zipf's law. *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference*, 2:727–730, 2000.

[32] Netflix. Netflix prize. `http://www.netflixprize.com/`, 2006. Online; accessed 09-Nov-2016.

[33] Meeyoung Cha Meeyoung Cha, Haewoon Kwak Haewoon Kwak, P. Rodriguez, Yong-Yeol Ahn Yong-Yeol Ahn, and Sue Moon Sue Moon. Analyzing the Video Popularity Characteristics of Large-Scale User Generated Content Systems. *IEEE/ACM Transactions on Networking*, 17, 2009.

[34] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2004.

[35] Ge Ma, Zhi Wang, Miao Zhang, Jiahui Ye, Minghua Chen, and Wenwu Zhu. Understanding Performance of Edge Content Caching for Mobile Video Streaming. *IEEE Journal on Selected Areas in Communications*, 35(5):1076–1089, 2017.

[36] Haitao Li, Lili Zhong, Jiangchuan Liu, Bo Li, and Ke Xu. Cost-Effective Partial Migration of VoD Services to Content Clouds. *2011 IEEE 4th International Conference on Cloud Computing*, pages 203–210, jul 2011.

[37] Zlatka Avramova, Sabine Wittevrongel, Herwig Bruneel, and Danny De Vleeschauwer. Analysis and modeling of video popularity evolution in various online video content systems: Power-law versus exponential decay. *1st International Conference on Evolving Internet, INTERNET 2009*, pages 95–100, 2009.

[38] Fei Chen, Haitao Li, and Jiangchuan Liu. On the impact of popularity decays in peer-to-peer VoD systems. *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, pages 1–6, jun 2013.

[39] Liang Chen, Yipeng Zhou, and Dah Ming Chiu. A lifetime model of online video popularity. *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2014.

[40] Hubert Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.

[41] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37(4):181, 2007.

[42] Martin J. Reed. Traffic engineering for information-centric networks. *2012 IEEE International Conference on Communications (ICC)*, pages 2660–2665, jun 2012.

[43] Lorenzo Saino, Ioannis Psara, and George Pavlou. Icarus: a Caching Simulator for Information Centric Networking (ICN). *7th International ICST Conference on Simulation Tools and Techniques*, 2014.

[44] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A Survey of Information-Centric Networking. 16(2):1–25, 2014.

[45] Nikos Fotiou and George C Polyzos. Broadband Communications, Networks, and Systems. 66(October):0–13, 2012.

[46] Dirk Trossen, Mikko Sarela, and Karen Sollins. Arguments for an information-centric internetworking architecture. *ACM SIGCOMM Computer Communication Review*, 40(2):26, 2010.

[47] Bander A. Alzahrani, Martin J. Reed, and Vassilios G. Vassilakis. Resistance against brute-force attacks on stateless forwarding in information centric networking. *ANCS 2015 - 11th 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 193–194, 2015.

[48] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. Cache Less for More in Information-Centric Networks. *Networking 2012*, 7289:27–40, 2012.

[49] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7):609–634, jul 2006.

[50] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic in-network caching for information-centric networks. *Proceedings of the second edition of the ICN workshop on Information-centric networking - ICN '12*, page 55, 2012.

[51] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Hash-routing schemes for information centric networking. *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking - ICN '13*, page 27, 2013.

[52] Lada A. Adamic1 Bernardo A. Huberman. Zipf 's law and the Internet. *Zipf's law and the Internet*, pages 143–150, 2002.

[53] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. On the implications of Zipf's law for web caching. Technical report, 1998.

[54] Kianoosh Mokhtarian and Hans-Arno Jacobsen. Caching in Video CDNs: Building Strong Lines of Defense. *Proceedings of the Ninth European Conference on Computer Systems*, pages 13:1—-13:13, 2014.

[55] Cedric Westphal, Tommaso Melodia, Wenwu Zhu, and Christian Timmerer. Guest Editorial Video Distribution over Future Internet. *IEEE Journal on Selected Areas in Communications*, 34(8):2061–2062, 2016.

[56] Benoit Mandelbrot. An informational theory of the statistical structure of language. *Communication Theory*, 84:486–502, 1953.

[57] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12, 2009.

[58] Mohamed Hefeeda and Osama Saleh. Traffic modeling and proportional partial caching for peer-to-peer systems. *IEEE/ACM Transactions on Networking*, 16(6):1447–1460, 2008.

[59] George Kingsley Zipf. Human behavior and the principle of least effort. *Addison-Wesley Press*, page 573, 1949.

[60] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[61] Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. *Vision research*, 49(10):1295–1306, 2009.

[62] A. A. Fedotov, P. Harremoes, and F. Topsoe. Refinements of Pinsker's inequality. *IEEE Transactions on Information Theory*, 49(6):1491–1498, June 2003.

[63] Frank Hansen and Gert K. Pedersen. Jensen's operator inequality. *Bulletin of the London Mathematical Society*, 35(4):553–564, 2003.

[64] Guillaume Pierre and Maarten Van Steen. Globule: A collaborative content delivery network. *IEEE Communications Magazine*, 44(8):127–133, 2006.

[65] B S Michel, K Nikoloudakis, P Reiher, and Zhang Lixia. URL forwarding and compression in adaptive Web caching. *INFOCOM 2000. Nineteenth Annual Joint*

Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 2(c):670–678 vol.2, 2000.

[66] Geant. `http://www.geant.org/`, 2000. [Online; accessed 09-Nov-2016].

[67] GARR. `http://www.garr.it/`, 2001. Online; accessed 09-Nov-2016.

[68] Tiscali. `http://www.tiscali.com/`, 1998. Online; accessed 09-Nov-2016.

[69] Jun Murai. Wide project. `http://two.wide.ad.jp/`, 1985. Online; accessed 09-Nov-2016.

[70] Henrik Abrahamsson and Mats Bjorkman. Caching for IPTV distribution with time-shift. *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 916–921, jan 2013.

[71] Henrik Abrahamsson and M. Bjorkman. Simulation of IPTV caching strategies. *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2010 International Symposium on*, pages 187–193, 2010.

[72] Youtube. `https://www.youtube.com/`, 2017.

[73] Bbc iplayer. `https://www.bbc.co.uk/iplayer/help/programme-availability/programme-availability-info`, 2017.

[74] Itv. `http://www.itv.com/help/the-itv-hub-help-videos/`, 2017.

[75] Channel 4. `http://www.channel4.com/devices/faq/ios.html`, 2017.

[76] Channel 5. `https://help.channel5.com/hc/en-gb/articles/200859561-How-long-are-shows-available-for-`, 2017.

[77] Sky. `https://www.sky.com/help/articles/using-sky-tv-on-demand`, 2017.

[78] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of Web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[79] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Edward A. Fox, and Stephen Williams. Removal policies in network caches for World-Wide Web documents. *ACM SIGCOMM Computer Communication Review*, 26(4):293–305, 1996.

[80] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, (December):18, 1997.

[81] a. Balamash and M. Krunz. An overview of web caching replacement algorithms. *IEEE Communications Surveys & Tutorials*, 6(2):44–56, 2004.

[82] Jia Wang. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36, oct 1999.

[83] Nimrod Megiddo and Dharmendra S. Modha. Outperforming LRU with an adaptive replacement cache algorithm. *Computer*, 37(4):58–65, 2004.

[84] Donghee Lee, Jongmoo Choi, Jong Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, 2001.

[85] Ioannis Psaras, Richard G. Clegg, Raul Landa, Wei Koong Chai, and George Pavlou. Modelling and evaluation of CCN-caching trees. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6640 LNCS, pages 78–91, 2011.

[86] David Holmes. The DDoS Threat Spectrum. 2016.

[87] Hazem Gomaa, Student Member, Geoffrey G Messier, Carey Williamson, and Robert Davies. Estimating Instantaneous Cache Hit Ratio Using Markov Chain Analysis. *IEEE/ACM Transactions on Networking*, pages 1–12, 2012.