Peer Assisted Multicast Streaming for On-Demand Applications

John Paul O'Neill

Trinity College Dublin, the University of Dublin

School of Computer Science and Statistics

A thesis submitted for the degree of

Doctor of Philosophy in Computer Science

March 2016

Declaration

Signature

John Paul O'Neill

March 2016

Acknowledgements

I would like to thank my supervisor, Dr Jonathan Dukes, for his guidance and support throughout my time working with him. I would also like to thank the other members of the Computer Architecture Group and the School of Computer Science and Statistics, in particular John Walsh for his technical support with my simulations.

The work described here has been part-funded by Irish Research Council for Science, Engineering & Technology and I am grateful for their support.

I want to thank Sarah for her continual support throughout, and my friends, especially Niall, Darren and Colin for providing an outlet for any frustrations. Finally, I would not have been able to finish this thesis without the support and encouragement of my parents Frances and John, and my siblings Stan, Stephen and Emma.

Abstract

On-demand multimedia streaming applications allow users to select and view content at a time of their choosing and to control the playback of this content. This level of control typically requires that each user receives a dedicated stream from the service provider. The provision of these streams can place a high load on a centralized server and can result in network bottlenecks near the server.

An approach called Peer Assisted Multicast Streaming is proposed to reduce the load on a centralized server, and to partially re-distribute the remaining load to the edges of the network.

Peer Assisted Multicast achieves this by allowing participating users to share with each other the beginning, or "prefix" of content items that they have previously received and cached, thus creating a decentralized collaborative cache. By doing this, a server will only need to serve the remainder of the content and can delay doing so until the content is required. This content can be provided using a multicast stream. Other requests received before the multicast stream begins can be serviced by the same stream. This allows the server to benefit from the use of multicast without affecting an individual user's experience. Serving multiple requests with a single stream will reduce the overall network bandwidth requirements of a server. As the users of the system will be providing content to each other, some of the network load will also be distributed away from the server.

Peer Assisted Multicast Streaming can be implemented using standard network protocols and existing peer-to-peer architectures. An implementation using the RTSP stream control protocol and the Chord overlay network is proposed.

Using a detailed network simulation Peer Assisted Multicast Streaming was evaluated under varying conditions. In the evaluation, Peer Assisted Multicast Streaming was shown to provide a 21.62% reduction in the number of concurrent multimedia streams required by the server when a prefix of 16.67% of the duration of the content was provided by the users. It was established that for the most popular content the maximum number of concurrent streams was independent of the request rate of the system, which is not the case for unicast based approaches. Peer Assisted Multicast Streaming was also shown to respond well to the sudden introduction of new popular content.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

On-demand, Internet-based multimedia streaming services are gaining mass-market acceptance as an alternative to traditional broadcast services for television and radio [Gro10]. The emergence and popularity of services such as BBC's iPlayer [1], Hulu [2] and NetFlix [3] is evidence of this. Services such as these, which had been used primarily by technology enthusiasts using desktop PCs, are now readily available to consumers through easy-to-use televisions, set-top boxes, video game consoles and smart phones.

Multimedia streaming services may be broadly classified into two groups, which are referred to here as *broadcast* services and *on-demand* services. There is a close relationship between the categorization of a service and the technology used to to deliver it.

In a broadcast service, the service provider controls the content that clients receive. Broadcast services are built around content schedules determined by the service provider. Service providers may offer a number of broadcast channels. Clients may choose to receive one of these channels, but have no further control over the broadcast content. All of the clients receiving the same channel will receive the same content at the same time. This broadcast approach is an efficient way to deliver either live or pre-recorded content and is the model used to deliver familiar services such as digital

[1]`http://www.bbc.co.uk/iplayer`
[2]`http://www.hulu.com`
[3]`http://www.netflix.com`

satellite television or Internet radio.

On-demand multimedia streaming services, in contrast, offer clients greater control over the content they receive. These services allow clients to select and view content at a time of their choosing and to control the playback of the content, for example, allowing clients to pause the playback of content, or to skip ahead to a future point of interest within the content. In other words, each client creates his or her own content schedule. This level of control prohibits the use of broadcast technology for offering on-demand services and makes on-demand multimedia streaming a resource-intensive application.

As the availability of high bandwidth Internet access becomes more widespread, the availability and popularity of on-demand streaming services has increased. Many content providers now offer on-demand libraries of content, either through their own websites in the form of so-called "catchup services" (e.g. BBC iPlayer or RTÉ Player [4]) or through dedicated on-demand streaming services (e.g. Hulu or NetFlix). Current trends indicate that on-demand services such as these are replacing traditional broadcast services [Gro10].

Offering on-demand streaming services to large numbers of clients is challenging and expensive, however, as each client requires an individual multimedia stream over which they have independent control. Furthermore, as the demand for high definition video and high quality audio increases, the demands placed on existing network infrastructure become a concern [Wak08]. The cost of delivering a broadcast service is usually independent of the number of clients and dependent only on the number of broadcast channels, and the quality of the stream. In contrast, the cost of delivering an on-demand streaming service is dependent on the number clients receiving the content as well as the quality of the stream. This is an important consideration for on-demand content providers as the cost of delivering the content is beyond their control and will be subject to the popularity of the service.

The focus of this thesis is on reducing the cost of delivering on-demand multimedia

[4] `http://www.rte.ie/player`

streaming services. The approach that is taken seeks to reduce this cost by reducing the number of individual streams required to deliver the on-demand service and also to shift the provision of part of some of these streams towards the edges of the network away from the server. This will be achieved without any reduction in the interactive functionality provided to clients.

In the proposed approach, which will be referred to as Peer Assisted Multicast Streaming or PAMS, each client maintains a local cache of the beginning, or "prefix", of each content item that they receive. Each client contributes the prefixes in its local cache, creating a "collaborative cache" from which a participating client can retrieve the prefix of a content item that it has requested. If a prefix can be obtained from the collaborative cache, the server only needs to supply the client with the remainder of the content item. The client does not need to begin receiving the remainder of the content until it has received the prefix. This provides an opportunity to the server to delay delivering the remainder of the content item. During this delay, if the server receives further requests for this content item, all of these delayed requests can be served using a single multicast stream. Multicast is a network delivery technique where one packet is sent, then duplicated and delivered to multiple destinations throughout the network. Multicast can offer a significant reduction in bandwidth utilisation for a server in the case where the same packets must be sent from a source to a number of destinations.

Both multicast and collaborative caching exploit the locality principle [Den05] to reduce the costs of supplying popular content. With multicast streaming, the locality principal is exploited by using a single multicast stream to serve clients who wish to receive the same portion of a content item at approximately the same time. Collaborative caching exploits the locality principle by allowing clients at a distance from the server to supply each other with recently received content items, thus reducing the load at and near the server.

By combining collaborative caching with multicast, Peer Assisted Multicast Streaming achieves the dual effect of reducing overall the number of streams required to deliver content items and shifting some of the cost of delivering those streams to the edges of the network. This is achieved without the introduction of additional dedicated infrastructure. Furthermore, clients retain the control that would normally be expected from an on-demand streaming service.

As the collaborative cache is implemented using the resources of the clients that access it, it will grow in proportion to the popularity of the system, thus providing a scalable solution for providing prefixes. Since the server still operates independently of the collaborative cache, the performance of the system will degrade gracefully towards that of a centralized unicast server if the performance of the collaborative cache deteriorates.

The collaborative cache uses a peer-to-peer approach to allow clients to locate prefixes stored by other clients. Peer-to-peer approaches for on-demand multimedia streaming have often been seen as unattractive by content providers, mainly due to concerns over Digital Rights Management and security. Since the approach proposed here, however, only uses the peer-to-peer network to provide a portion of each content item, the access to the reminder of the content can still be restricted by the content provider at the server.

As the collaborative cache is independent of the server, it could also be used in conjunction with a non-multicast server to reduce bandwidth utilisation of the server, without the use of multicast, by caching and serving prefixes in a similar manner to the proxy prefix cache architecture proposed elsewhere [SRT99].

A number of challenges need to be addressed by PAMS. Any system based on the use of peer-to-peer technology will be subjected to the effects of churn, as existing peers leave the network and new peers join it. The consequences of churn must be considered, measured and, where necessary addressed. Furthermore, PAMS must be able to adapt quickly to the introduction of new content or to the variation in the popularity of content. In particular, PAMS must be capable of adapting quickly to the occurrence of so-called “flash crowds” [JKR02], where a significant and unpredicted increase in usage of the system or the accesses for a particular content item within the system occurs.

In implementing PAMS, some key philosophies dictated the approach taken and decisions made. It was felt that the collaborative cache should be lightweight, with regard to the network overhead required to maintain the collaborative cache. It was also decided that servers should act independently of the cache and that a server should require only minor modifications to implement PAMS. The implementation of PAMS should also ensure that the service is not dependent on the collaborative cache but is augmented by it. Finally, the design of PAMS should support client and server implementations that are compatible with standard network streaming protocols.

Experiments in Chapter 7 will demonstrate that PAMS can significantly reduce outgoing server bandwidth utilization in comparison to the bandwidth required by a simple unicast approach. Furthermore it will be be shown that PAMS scales more effectively than a simple unicast approach as the number of clients accessing the service increases. In particular, unlike a unicast approach, when the number of clients requesting a particularly popular content item is sufficiently high the server resources required to deliver this content ceases to increase. In this situation, this maximum limit is independent of the popularity of the content item. This has important implications for resource provisioning for a service provider. It will also be shown that PAMS adapts effectively to sudden increases in the popularity of content.

Alternative approaches to the one proposed here typically focus on either reducing the number of streams required to serve on-demand content, or distributing the load to other locations in the network. Multicast is one way of reducing the total number of streams required to deliver an on-demand service. The challenge when using multicast, however, is to achieve this reduction without losing the interactivity that clients expect from an on-demand service.

Instead of reducing the number of streams required to provide content, Content Delivery Networks (CDNs) seek to distribute the network load associated with the on-demand service to other locations in the network and, in particular, locations that are closer to the clients receiving the on-demand streams. This is achieved by deploying additional servers and replicating the content. CDNs are expensive, however, due to the cost of deploying additional server infrastructure.

A more cost-effective approach to distributing the load generated by an on-demand service is to use a peer-to-peer based system. Peer-to-peer systems adapt well to increases and decreases in usage, but without a dedicated centralized server from which all content can be retrieved, some content may become unavailable within the peer-to-peer system.

There are some issues which fall outside the scope of this thesis. For example, issues related to digital rights management and security have not been considered. Providing a Quality of Service guarantee to consumers of the streamed content is also outside of the remit of this thesis.

Contributions

The primary contribution of this thesis is the Peer Assisted Multicast Streaming (PAMS) architecture. The architecture represents a new approach for delivering on-demand multimedia content. The approach combines the use of multicast with a peer-to-peer collaborative cache. In addition to the design of the architecture an approach for implementing the PAMS using standard network protocols and existing multicast and peer-to-peer overlay technologies is proposed, and aspects of the approach may be applicable to other multicast streaming architectures.

A detailed simulation framework for evaluating large-scale multimedia systems was developed. The framework simulates behaviour of multimedia streaming systems down to the network packet level. This framework was used to perform a detailed performance evaluation of Peer Assisted Multicast Streaming. Furthermore, the framework was used in the re-evaluation of existing multimedia approaches.

Finally, the thesis presents a state of the art review of multicast and peer-to-peer on-demand streaming systems.

Overview

Chapter 2 provides an overview of multimedia streaming services, with an emphasis on the properties of on-demand systems and on the approaches to their implementation. The two subsequent chapters, Chapter 3 and Chapter 4, provide an introduction to multicast and peer-to-peer technologies, as well as describing their use in multimedia streaming applications. In Chapter 5, the proposed Peer Assisted Multicast Streaming architecture is described in detail. A proposed approach to the implementation of PAMS is described in Chapter 6, along with a description of how the behaviour and performance of PAMS has been evaluated. The results of this evaluation are then presented and discussed in Chapter 7. Finally, conclusions and proposed areas of future work are presented in Chapter 8.

Chapter 2

On-Demand Multimedia Streaming

In this chapter, the nature, properties and requirements of on-demand multimedia streaming systems will be presented and discussed. An initial description of what is meant by the term "multimedia" will serve to define more precisely the context of the thesis. This will be followed by a discussion of multimedia streaming applications, in which multimedia content is transmitted between network nodes. The properties and requirements of different types of multimedia streaming applications, will be presented and compared, focusing on the on-demand applications that are the main concern of this thesis. Finally, different approaches to the efficient implementation of on-demand multimedia streaming services will be presented and discussed.

2.1 Multimedia Systems

Multimedia systems are concerned with the capture or creation of content, such as audio or video, in digital form, its storage in digital form and its subsequent processing, retrieval and rendering. Today, the term "multimedia" usually implies either audio or video content or, frequently, a combination of the two, but may also include other media such as image galleries, subtitles, closed-captions and embedded links to other content.

In analogue form, video is stored on film as a series of images, often referred to as frames. These frames, when displayed, or "*rendered*", in rapid succession appear to a viewer to be a moving image. The speed at which the images are displayed is referred to as the frame rate, and is measured in frames per second (fps). The usual rate of an analogue cinema presentation is 24 frames per second.

This approach can also be applied to digitizing video, where each image is digitally stored and then rendered at a specified framerate. A single digital image is made up of a collection of pixels, where each pixel represents a colour. The number of bits used to represent a the colour of a pixel is referred to as the colour depth. To represent two colours (e.g. black and white), a single bit can be used. The larger the number of bits used, the more colours are available, with 24 bits (16,777,216 colours) being an acceptable standard for photo-realistic image quality. The number of pixels used to represent an image can vary from one format to another and is known as the resolution. A typical standard of resolution is 720 horizontal pixels and 480 vertical pixels.

The basic storage requirements of an image can be calculated as the product of the number of pixels (resolution) and the number of bits used to represent the colour of each pixel (colour depth). For example, the basic storage requirement of a digital image using the resolution and colour depth described above would be $720 \times 480 \times 24 = 8294400$ bits or 0.99 MB.

Techniques for compressing digital images such as JPEG, PNG and GIF are frequently used to reduce the amount of space required to store a digital image, but these will be ignored for this example.

The frame rate combined with the resolution and colour depth determines the bit rate of uncompressed video. For example, a frame rate of 24fps, with the above resolution would require a bitrate of approximately 199Mbps (Megabits per second). Figure 2.1 illustrates how frame rate and resolution are combined to produce a bit-rate.

If a ninety minute movie was to be stored in an uncompressed format, this would require $90 \times 60 \times 24 = 129600$ frames to be stored. This would require approximately

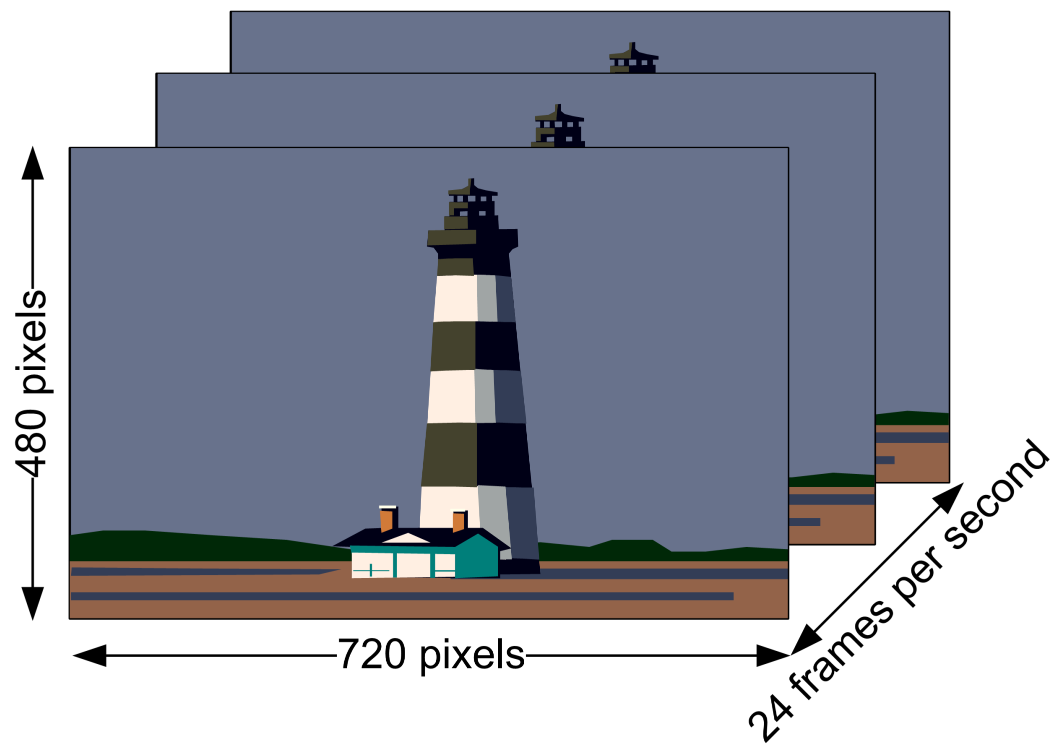


Figure 2.1: Storage requirement for uncompressed video

125GB of storage, which is clearly unfeasible.

To overcome these storage requirements, video compression standards are used, such as MPEG-4 [Koe99] and H.264 [WSBL03]. These compression algorithms exploit the similarities between consecutive frames. The differences between two frames are generally minuscule, so, to save space, rather than save two consecutive frames, these algorithms are based around storing the differences between the current frame and the following frame. Using MPEG-2 compression, which is the standard upon which DVD's are based, a resolution of 720×480 pixels, with a 24 bit colour depth and a frame rate of 30fps will result in a bit rate of approximately 9.8Mbps. Assuming this bit rate, the storage requirements for a ninety minute ($90 \times 60 = 5400$ second) DVD quality video would be:

$$5400 \times 9.8 \approx 5.29 \times 10^{10} \text{ bits} \approx 6.16\text{GB}$$

Multimedia presentations typically contain audio as well as video. Audio is normally digitised using pulse-code modulation. Pulse-code modulation requires the amplitude of the audio to be sampled at specific intervals and this is known as the sampling rate. The amplitude of the audio at the sampling time is quantized to a discrete value determined by the bit depth and the quantization algorithm. A higher bit depth and sampling rate give rise to better quality recordings of the original audio, but this will result in higher storage requirements. Digitised audio can be compressed using algorithms such as MP3 [Bra99].

Using linear pulse code modulation, the amplitude of an audio wave is sampled at a constant rate and quantized to a discrete value, with the number of possible values determined by the bit-depth. For example, 8-bit audio provides 2^8 unique values to represent the sampled amplitude. CD-quality audio has a sampling rate of 44,100Hz and a bit-depth of 16, meaning that the amplitude of the original audio wave was sampled 44,100 times per second and each sample was assigned a 16-bit value.

To allow multiple media types to be stored together, for example audio and video in a movie, media containers are used. Some commonly used media containers are AVI, MP4, FLV and MPEG-TS, with FLV being commonly used for embedded video on websites (e.g. YouTube, Hulu) and MPEG-TS used in satellite digital video broadcasting (DVB-S). To display the multimedia content correctly, the media streams must be demultiplexed into their constituent streams and then rendered independently but with synchronization between the streams (e.g. to ensure "lip-sync" is maintained).

Encoded multimedia content may be retrieved either from local storage or from a remote source. Locally stored multimedia content would be accessed from a hard drive, or from optical media such as a DVD, from where it can be accessed and rendered at a constant bit-rate. Remotely accessed content though, would need to be delivered over a network from a multimedia server or via a broadcast medium such as digital satellite. When delivered via a network, a request is sent for the content from the client to a server and the server will then deliver the content to the client. This is considered to be remote retrieval. Remote retrieval offers users the ability to watch a variety of content types such as news items, films or sporting events. There are two models for remote retrieval: download and streaming.

In a download based system, the entirety of content must be retrieved from the server by the client and stored locally before rendering of the content can begin. Once the content has been fully retrieved it is accessed and rendered in the same way as locally stored content would be rendered. The single biggest disadvantage of a download based system is that it can take a substantial amount of time to fully download content, with the time required determined by the bitrate of the content, the available download bandwidth and the duration of the content. Download based services also require the client to have sufficient storage space to store the entirety of the content.

A more advanced model of download is progressive download, in which playback of content can be started once download has commenced. This significantly reduces the time a requesting client must wait to begin consuming the requested content, but has the disadvantage of reducing the interactivity the client can have within playback as features such as skipping to a future point in the content item are not supported.

In a multimedia streaming application, a multimedia stream is transmitted from a source location, over transport medium such as a network, to a destination, where it is rendered, processed or stored as it is received. The transmitted content may be pre-existing, stored content (e.g. a video-on-demand service) or it may be produced in real-time (e.g. a video conference or live news broadcast).

In a streaming based service, rendering can begin once the client has received the start of the content. Streaming services require less client storage capacity than download services, as once the content has been rendered it can be discarded. The duration of the content is a factor in determining the storage space required in a download system, but in a streaming system, the duration of the content has little impact on the client's required client storage capacity as content does not need to be stored once it has been rendered. A small amount of storage is required to buffer content as it is received, this is discussed below.

From a server's perspective, a streaming service is more demanding in terms of

Property	**Download Service**	**Streaming Service**
Client Latency	High - content must be fully received before playback can commence	Low - playback can begin once buffer has received enough content
Server Load	Server can balance needs of all requests as there are no real-time constraints	Server must deliver content to each requesting client in a timely fashion to avoid clients experiencing interruption in playback
Client Storage	Clients must have sufficient storage to store entirety of content	Clients need only have enough storage to meet the buffers requirements
Playback	Playback is only started once all content is received, so will be uninterrupted	Playback is subject to interruption, due to changes in network quality, a buffer is required to overcome this
Commercial Examples	iTunes, Amazon	YouTube, BBC iPlayer, Hulu

Table 2.1: Comparison between Download and Streaming Services

resources. In a download system the server can deliver the content at a rate that is limited by the bandwidth available between the server and the client. In a streaming service though, as the client has real-time demands, clients must receive the content stream at a pre-determined rate, determined by the encoding of the content. If the client receives the content too slowly then rendering of the content will be effected, and the client will not see an uninterrupted media presentation. In contrast, if the client receives the content too quickly, it will need to buffer the content before rendering it, which increases the local storage requirements of the client. If the server can deliver the content at a rate that approximately matches the bitrate of the content, the client may still need to maintain a small buffer to allow for variations in the rate at which content is received (for example, as a result of network congestion) or the availability of local resources to render the content. The properties of download and streaming services are summarised in Table 2.1

2.2 Multimedia Streaming Applications

Multimedia streaming applications can be broadly characterised as “broadcast” or “on-demand” applications, depending on the degree of control that clients have over the streams they receive. In this section, the characteristics of broadcast and on-demand streaming applications will be compared while the following section will discuss the resource requirements of both.

2.2.1 Broadcast Streaming Applications

In a broadcast streaming application, the content that is transmitted and the time that it is transmitted at is determined at the source of the transmission. Clients are limited to choosing whether or not a transmitted stream is received. The content streams may originate from an archive, for example, in the case of broadcast television. Alternatively, the content streams may be produced in real-time, for example, in the case of a live event broadcast over the Internet or a video conference

Broadcast streaming can be delivered over a broadcast medium, such as digital satellite, or it can be delivered over a network through a stream from a server to a number of clients, with each client receiving the same data at approximately the same time. Broadcast streaming services do not offer clients the ability to interact with streams, i.e. clients cannot view a different part of the broadcast than is currently being provided by the server.

2.2.2 On-Demand Streaming Applications

In contrast, in an on-demand streaming application, the client chooses what content should be transmitted by the source of the stream from an archive of stored content. Furthermore, users can interact with a stream in the same way that they might interact with the playback of a DVD (e.g. by pausing, rewinding or jumping to a different point in the stream). Due to server resource limitations though, a majority of video on demand streaming service only allow playback of content at the normal rate, with fast forward and rewind only being possible on buffered parts of the content as these operations would require the client to receive the content at a higher than normal bitrate, which may exceed the available server's, network's or client's available bandwidth. To counter this limitation, services allow playback of content to begin from any point of the content, which allows the user to skip to any part of the content they wish to view.

Examples of on-demand streaming applications include TV catchup or movie-on-demand services.

2.3 Resource Requirements of Streaming Applications

The characteristics and, in particular, resource requirements of a multimedia streaming application depend on whether the service offered is broadcast or on-demand and these characteristics and requirements are discussed here.

The main parameters that determine the resource requirements of a streaming application can be divided into two types, requirements that result from the characteristics of the content being served and requirements that result from the nature of client demands.

The bitrate and duration of content will determine the requirements of a stream served to a client, while the total number of concurrent clients and the method of delivery will determine the requirements for the system as a whole.

2.3.1 Broadcast Streaming Server Resources

Resource requirements of a broadcast streaming application are dependent on how the broadcast service is implemented. Broadcast streaming applications can be delivered using broadcast, unicast or multicast delivery. With both broadcast and multicast delivery, the server will only deliver a single multimedia stream for each channel which all clients will receive. Broadcast techniques such as network-level broadcast involve data being delivered to all receivers, whereas multicast involves a single data packet being delivered to a select group of receivers.

The server resources for Broadcast and Multicast are determined by the number of content items provided by the server and the properties of those content items including, in particular, the bitrate. The server resource requirements in a broadcast or multicast system are independent of the number of clients. Broadcast can be used over a medium such as digital satellite, while multicast can be used to deliver content over networks such as the Internet. Multicast delivery is discussed in further detail in Chapter 3.

A unicast approach may also be used to implement live streaming applications, in circumstances where broadcast or multicast delivery is not available. Unicast, however, has significantly higher server resource requirements than either broadcast or multicast approaches, since the server must supply an individual stream to each client. This results in the resource requirements of the server being determined by the number of concurrent users as well as the properties of the stream.

2.3.2 On-Demand Streaming Server Resources

The focus of this thesis is on on-demand multimedia streaming applications and, in particular, on reducing the server and network resource requirements of such applications. Basic on-demand streaming services provide each user with an individual data stream, placing a substantial demand on the resources of service providers. The resource requirements of a multimedia streaming service are determined by the number of clients accessing the service, the duration of each concurrent stream, the bitrate of each stream and the total number of content items offered by the service. Due to decreasing cost of storage, storage space is no longer a determining factor for how many content items are offered by the service. The bandwidth available to a streaming service is a major factor though, as multimedia streams tend to be both bandwidth intensive and long-lived.

For a service offering N content items, each with a known bit rate, B_i, and an average stream duration, Lt_i, then the mean bit rate for all concurrent streams in the system can be calculated for a known mean request rate of R. For a content item i which is requested with probability p_i, the request rate for the item i is $R \times p_i$. Using Little's formula [Lit61][1], which may be rewritten in this context as $S_i = R \times p_i \times Lt_i$, the number of concurrent streams of a content item (S_i) can be calculated for a given request rate and mean stream duration. For content item i, the mean number of concurrent streams is $R \times p_i \times Lt_i$ and the bandwidth required for this number of concurrent streams is

[1] $L = \lambda W$, where W is the mean duration of a request, λ is the mean arrival rate of requests, and L is the mean number of concurrent requests in the system

$R \times p_i \times Lt_i \times B_i$. Thus, the aggregate bandwidth required to serve all content items is

$$R \times \left(\sum_{i=1}^{N} (p_i \times Lt_i \times B_i) \right)$$

To illustrate this, consider a service that receives a client request every 4 seconds. Each client requests a movie with an average duration of 3600 seconds and each stream has a bandwidth requirement of 5Mbps. In this example, the total bandwidth requirement of the service is 4,500Mbps.

Although this simple example illustrates how factors such as bit-rate and stream duration impact on the bandwidth required to supply a multimedia streaming service, there are other factors that must also be considered.

Concurrent Users While the calculation above gives the mean number of concurrent users a service could handle, the actual number of concurrent users at a given time can vary significantly. Factors that can cause this include the time of day, with a system experiencing peak loads at certain times. In extreme cases, "flash crowds" [JKR02] may occur, where there is a substantial and unpredicted increase in the load on the system. Such an event can occur during a breaking-news event or after the release of new and highly popular content and can result in the service becoming overloaded.

Popularity of Content The distribution of requests over the entire collection of content items can be modelled using a Zipf [Zip49] [SD00] distribution. For a simple streaming service, the popularity of individual titles has no affect on the requirements of a system, unless the content items have differing bit-rates and durations.

2.3.3 Client Resources

As discussed previously, streamed multimedia content must be delivered across a network according to a schedule determined by the bit-rate of the encoded content. The time required to transmit data from a server to the client is not constant and is subject

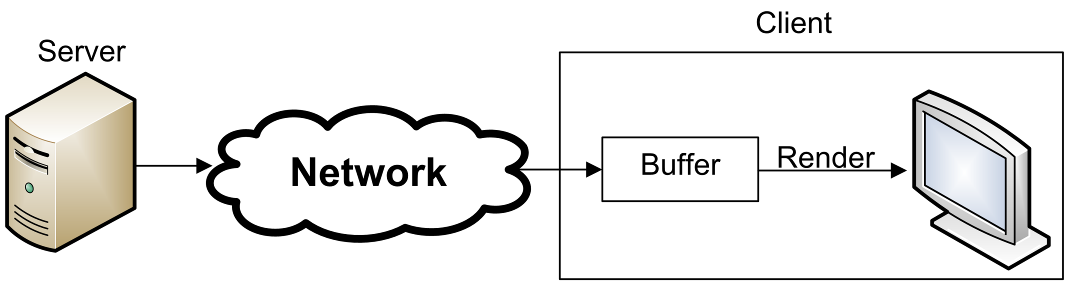


Figure 2.2: Simple buffer

to some fluctuation. Due to changing network conditions between the two endpoints, there may be some variation in the time required to transmit any two packets from the server to the client. This variation is commonly referred to as jitter, and is outside of the control of the endpoints. This results in fluctuations in the rate at which the media content is received, which could result in the content being received at a rate less than the bitrate of the content. If this was the case the result would be an interruption to rendering of the content. To prevent such fluctuations being perceived by users in the rendering of the content, received multimedia streams are usually buffered by the client.

Clients implement buffering by locally storing streamed content as it is received. The buffered content is then rendered from the buffer. Rendering will only begin once the buffer is filled sufficiently to overcome the expected jitter. This allows for continuous playback if there are delays in the transmission, as there is still multimedia content available to be rendered. A buffer not only allows for some delays in the transmission, but also allows for the content to be received at a higher rate than the bit-rate of the content. Depending on the size of the buffer, the entirety of the content could be received by the client long before it is due to be rendered.

The main disadvantage of a buffer is that it introduces a further delay before the client can begin rendering the multimedia stream, as this cannot start until the buffer is filled to a sufficient level. The more unreliable the network between client and server is, the larger the amount of initial content the buffer will need before the content can be reliably rendered at a constant rate. Figure 2.2 illustrates illustrates buffering.

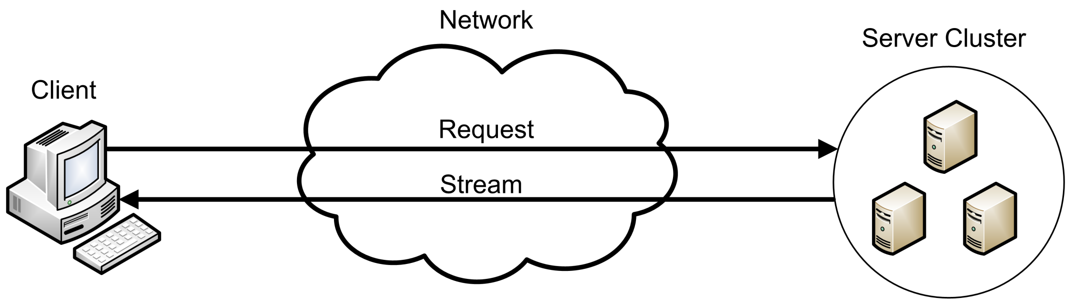


Figure 2.3: Centralised Content Delivery

2.4 On-Demand Multimedia Streaming Architectures

A number of techniques have been proposed in the literature and used in practice to provide on-demand streaming services. This section will discuss the main approaches that have been used commercially or proposed in the literature.

2.4.1 Centralised Content Delivery

The most basic approach for implementing a multimedia streaming service is a centralised architecture. In this architecture a single server or a centralised cluster of servers will receive the requests from the clients and will service these requests by providing streams containing the requested content.

2.4.2 Content Distribution Networks

The use of Content Distribution Networks (CDNs) has become commonplace for provision of large scale multimedia streaming services. CDNs distribute content from an origin server to replica servers, from which clients receive their content directly. This architecture has numerous advantages. Often the replica servers are closer to the clients at the edges of the network, which can avoid the need for multimedia content to traverse congested parts of a network, thus both improving service scalability and reducing latency. CDNs also reduce the load on the origin server as only the replica servers will contact the origin server.

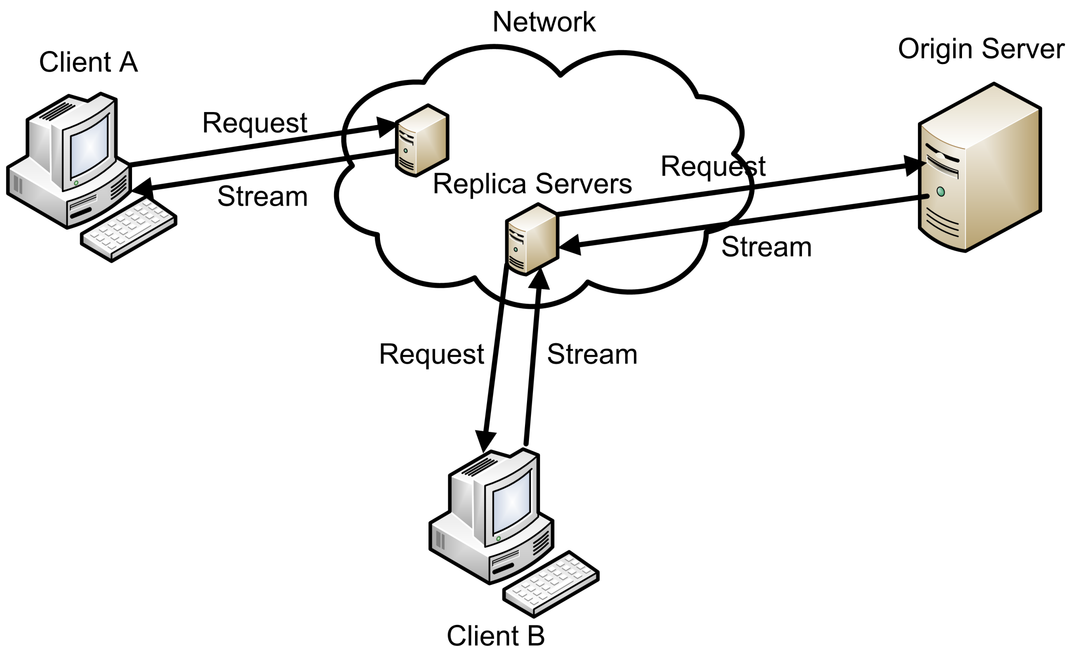


Figure 2.4: Content Delivery Network

2.4.3 Proxy Prefix Caching

Proxy prefix caches [SRT99] are similar in architecture to CDNs but instead of replicating entire content items, only the beginning of each stream is cached and served to requesting clients. The rest of the content is then streamed from the server to the client through the proxy prefix cache, with the cache responsible for smoothing the delivery of content to allow for a near constant bit rate supplied to the client. The effect of proxy prefix caching is that streams have low startup latency and the bit-rate of the received stream can be controlled as there is less likelihood of variable jitter between the prefix cache and client than there is between the server and client.

2.4.4 Peer-to-Peer Systems

Peer-to-peer systems are mainly used for live streaming as they offer a scalable solution to live streaming demands. For on-demand services, they are less used due to the volatility of their resources. Peer-to-peer multimedia systems are further discussed in Chapter 4.

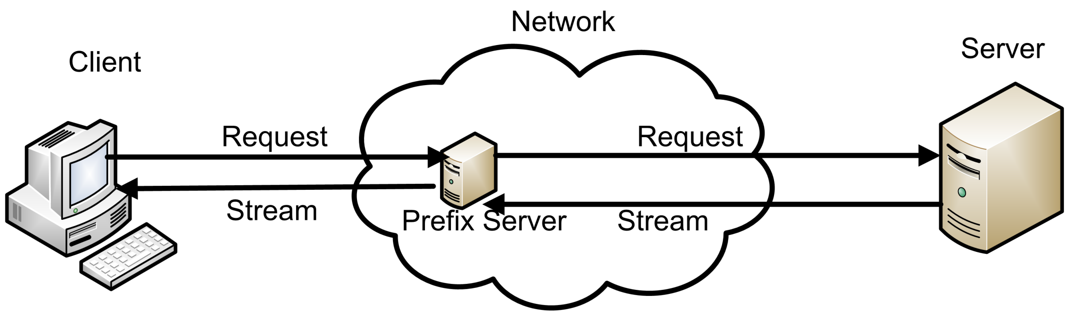


Figure 2.5: Proxy Prefix Caching

2.4.5 Comparison of On-Demand Multimedia Streaming Architectures

The main disadvantage of centralised content delivery architectures is their limited scalability. Although adding more resources to a centralised cluster may allow the cluster to provide more concurrent streams, network bandwidth remains a constraint on service capacity. Furthermore, the server is inherently a single point of failure in the architecture. Finally, if the network distance between a centralised server and a client is substantial, clients may experience high stream startup latencies between requesting a content item and rendering the beginning of the content.

Although Content Delivery Networks and Proxy Prefix Caches address issues such as startup latency, network congestion and server load, they both require the deployment of additional infrastructure. The costs involved in deploying and maintaining CDNs can be prohibitive and, unless there is a guaranteed demand on the network, the benefits can be outweighed by the costs. It is now common practice for multimedia content providers to outsource the provision of CDNs to dedicated providers, such as Akamai [2]. Currently, Akamai has over 95,000 servers in 71 countries operating within nearly 1,900 networks [3].

Flash crowds [JKR02], which are defined as an unexpected surge in accesses to a system, can also affect CDNs as they may be slow to adapt to changes in content popularity. Furthermore, decreasing popularity of content is difficult to define as these

[2]`http://www.akamai.com`
[3]`http://www.akamai.com/html/about/facts_figures.html`

decreases could be linked to other factors, such as time of day, rather than an actual decrease in the access rate of the content.

2.5 Implementing Multimedia Streaming

In order to establish, control and teardown a multimedia streaming session between two points, an appropriate protocols must be used. One such commonly used control protocol is the Real Time Streaming Protocol (RTSP) [SRL98]. RTSP is now discussed in detail as it is later proposed in Chapter 6 as a suitable protocol for controlling streaming in the Peer Assisted Multicast Streaming architecture proposed in this thesis.

The details of the operation of RTSP are described below. While RTSP can be used for a variety of multimedia streaming applications, including, for example, recording applications that may involve the transmission of a stream from a client to a server, the protocol will only be discussed in the context of the control of a stream from a server to a client and in the context of on-demand multimedia streaming applications in particular.

2.5.1 Real-Time Streaming Protocol

RTSP is an application level protocol in the OSI model [Tan96] for the control of multimedia streams. Typically there will be two parties involved in an RTSP session, the client requesting the stream and the server providing the stream. Within an RTSP session, both parties can issue requests that can effect the control of the streaming session. As RTSP is an application level protocol, the connection between the client and server will use a transport level protocol such as TCP [Pos81]. The multimedia stream itself is delivered out-of-band of with respect to the RTSP session through a different protocol such as RTP [SCFJ03] or UDP [Pos80].

RTSP is similar to the Hyper Text Transfer Protocol (HTTP) [FGM$^+$99] used to request and receive web content. RTSP messages are similar in format to HTTP messages and many messages are based on their HTTP counterparts. Unlike HTTP, however, the client and server in an RTSP session must maintain the state of the session.

2.5.2 RTSP Messages

RTSP messages are sent between a client and a server in order to control a stream. Each request message requires a response message to be sent. The format of messages is based on HTTP, with a header specifying the type of message, followed by the body of the message. Generally, a client will issue requests and the server will issue responses, although the server may also issue requests, such as those to change the transport parameters of the stream or to teardown a stream. Upon sending or receiving a request or response, a change of state can occur at either the client or the server, depending on the nature of the message.

Only a small subset of the functionality of RTSP must be considered in the specific context of on-demand streaming applications, which are the focus of this thesis. The following RTSP request types will be of interest:

`SETUP` Upon receipt of a `SETUP` request, the server will allocate resources for the stream, and create a new RTSP session. However, it is important to note that the server does not commence transmitting the multimedia content at this point. In a `SETUP` request, a client can specify the transport parameters that are available and acceptable to it. The server, in its response, will provide the client with details of the transport parameters that it has selected for the session. Upon receipt of the response the client can open up the required transport channel with the server.

`PLAY` After receiving the response to a `SETUP` request, a client may issue a `PLAY` request to the server. Upon receipt of this request, the server will respond to the request and, if the response is positive (`OK`), will begin sending the data through the agreed transport channel.

`PAUSE` While receiving a stream, a client may issue a `PAUSE` request to the server. If this request is successful, the server will respond positively (`OK`) and temporarily

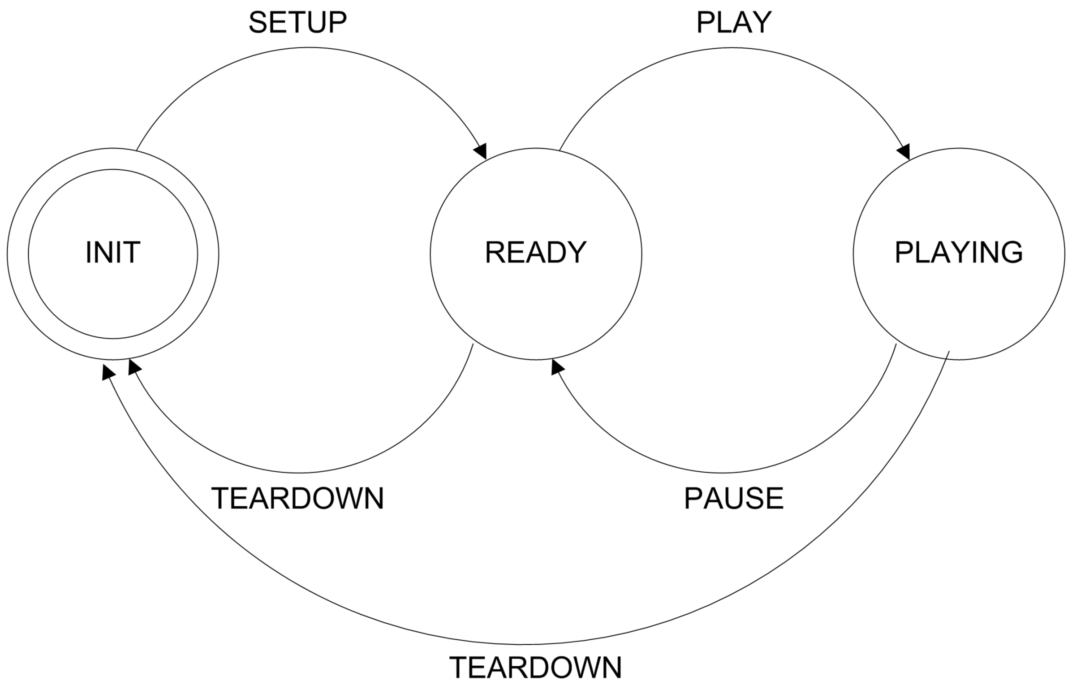


Figure 2.6: RTSP State Machine

suspend sending data to the client. If the client wishes to resume playback, it may issue a `PLAY` request again.

`TEARDOWN` A `TEARDOWN` message can be issued by either a server or a client. In either case, the server will free any allocated resources, and the RTSP session will be ended. `TEARDOWN` will be the only request sent by a server, with all other requests sent by clients.

2.5.3 RTSP Session and States

A simple RTSP session and the states of the client and server is outlined in Figure 2.6. Generally, a state change occurs in the client upon receipt of a response to a request while the server's state will change upon the receipt of a request.

In order to establish a multimedia stream, a client must send a `SETUP` request to the server. On sending the request, the client enters the INIT state, pending a response from the server. Before a `SETUP` is received by the server, it's state can be considered to be INIT as well. Upon receipt of the `SETUP` request, if the server has sufficient resources to provide the stream, a response will be sent with the agreed transport parameters and the server will enter the READY state. Upon receipt of the response, the client will enter the READY state as well.

In the READY state, all transport parameters for the stream have been agreed and the control of the playback of the stream can now be performed. In order to start receiving the content the client will issue a `PLAY` request. If the SERVER is able to begin streaming, the response to a `PLAY` request will be `OK`. In this case, the server will enter the PLAYING state and begin transmission of the content. On receipt of the response, the client will also enter its PLAYING state.

While in the PLAYING state, streaming can be temporarily suspended by the client by sending a `PAUSE` request. If successful, this will result in the server re-entering the READY state and, on receipt of the `OK` response, the client will also re-enter its READY state. From this state, a `PLAY` request can again be sent to the server to request the resumption of streaming. The server must record the point at which the stream was paused to allow the stream to be resumed at the same point.

To end a session completely, a `TEARDOWN` request must be sent by either the client or the server. The result of a `TEARDOWN` request will be to free up the transport channel resources.

2.6 Summary

This chapter began with an introduction to multimedia and multimedia streaming applications. Multimedia streaming applications may be classified as either broadcast or on-demand applications. In general, broadcast streaming applications are ideally suited to delivery using broadcast or multicast approaches, while on-demand applications usually require a unicast approach, making them significantly more server and network resource intensive.

This chapter has also discussed how on-demand multimedia streaming services are implemented using the RTSP protocol. RTSP will be proposed in Chapter 6 as a suitable protocol for the implementation of the Peer Assisted Multicast Streaming architecture proposed in this thesis.

Chapter 3

Multicast Multimedia Streaming

The Peer Assisted Multicast Streaming architecture proposed in this thesis combines multicast and peer-to-peer technologies to reduce the cost of providing on-demand streaming services. This chapter will begin with an overview of multicast and its implementation, followed by a description of how multicast can be used to provide multimedia services, and specifically on-demand streaming services.

3.1 Multicast

Multicast provides a means of sending the same data from a source to multiple destinations without requiring the source to send multiple copies of the data. The destinations receiving the same data are often referred to as a multicast group.

Multicast can be supported at the network level, through multicast enabled infrastructure, or in the absence of multicast enabled infrastructure, may be delivered through application level support. While network level multicast is more efficient and has lower associated overheads, the historical lack of support by Internet Service Providers (ISPs) means that application level multicast is often used as a means to delivering multicast traffic.

The general multicast model of data delivery can be described in three stages. A receiver expresses an interest in a multicast data stream by informing the multicast system of the identifier of the associated group, thereby joining the multicast group. A sender sends data to this multicast group, specifying the group's identifier. Finally, the multicast system will be responsible for routing the data to each member of the multicast group. An important consideration when implementing multicast delivery is establishing the multicast tree that will result in packets being delivered to the receivers in the multicast group.

The focus of the remainder of this chapter is on network level multicast as it has the potential to provide the greatest efficiencies.

Network Level Multicast

IP Multicast, as proposed by Deering et al. [DC90], allows a single IP packet to be delivered to multiple Internet hosts. This has the potential to significantly reduce network traffic generated by an application that requires the same data to be sent to multiple receivers, as the total number of packets required for data transmission is reduced.

For IPv4 networks, in order to establish interest in a multicast address by a receiver, the Internet Group Management Protocol (IGMP) [Dee89] is used. In IPv6 networks the Multicast Listener Discovery Protocol (MLD) [DFH99] is used. Various versions [Fen97] [CDK$^+$02] [VC04] of these protocols are in use, with different features, for example, relating to the manner in which routers communicate with each other and how the multicast group subscriptions of surrounding routers and clients are expressed.

MLD Version 2 [VC04] (MLDv2) will now be described, as this protocol allows *source specific multicast* to be implemented. Source specific multicast, which is described in Section 3.1.1, is ideally suited to on-demand multimedia streaming applications, which are the focus of this thesis, as the multicast content is delivered from a single centralised source.

Multicast Listener Discovery Version 2

The MLDv2 protocol allows routers to discover, for each direct link attached to the router, the multicast addresses, and optionally the associated source or sources of traffic sent to that address, that the entities connected to the link are interested in. This information allows routers to direct packets intended for these multicast addresses to the required links. The router may be required to duplicate and send the packet to zero, one or more connected links depending on the multicast group interests of each link.

There are two roles specified within the MLDv2 protocol, *multicast routers* and *multicast address listeners*. In practice, end-users (clients) of the network are multicast address listeners and routers act as both multicast routers and multicast address listeners.

The current interests of each link are maintained by using MLDv2 Query Messages. A multicast router will periodically send these messages to each of its links in order to learn and refresh the multicast group interests for each connected link. Upon receipt of a query, a multicast listener will respond to the multicast router with a MLD Current State Report listing the multicast addresses, and, if applicable the source address the listener is interested in receiving data from. In the case where a number of multicast routers exist within a subnet, one of the routers is elected to be the Querier, which is the router responsible for sending periodic query messages, the responses to which allow all the multicast routers to maintain the correct state of their routing tables. MLDv2 is a sub-protocol of ICMPv6 [CDG06], and both the Query and Report messages and are based on ICMPv6 messages.

The format of a Multicast Listener Query Message is shown in Figure 3.1. These messages allow the router to establish the interests of the multicast address listeners on its links. These query messages can be used in two ways. Firstly, if the Multicast Address field is zero, the multicast address listener must respond with all of its current multicast group interests. Otherwise if the multicast address and associated source address are specified, then multicast address listeners will only respond if the address

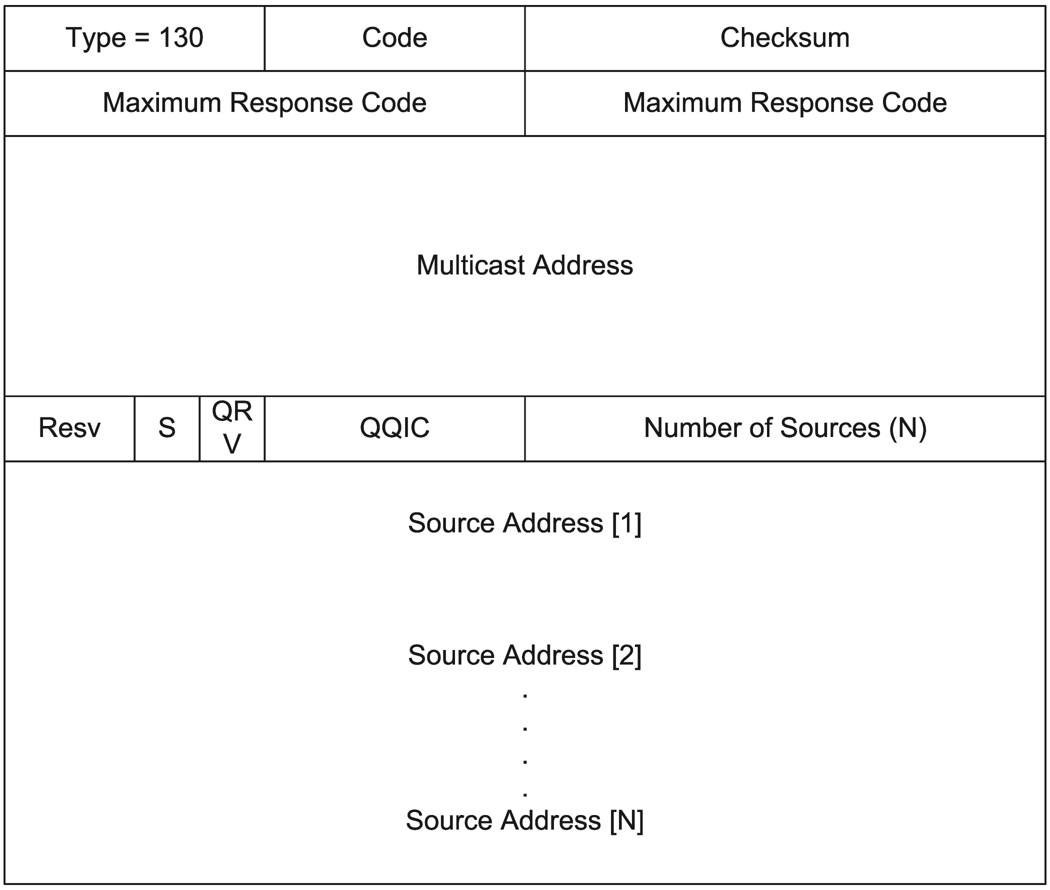


Figure 3.1: Multicast Listener Query Message [VC04]

matches the listeners' current interests, otherwise they will ignore it.

In order to respond to a query or to alert the multicast router to a change in interest in a multicast address, a listener will send a Multicast Listener Report to the multicast routers it is connected to. The format of a Multicast Listener Report is shown in Figure 3.2. The key component of the Multicast Listener Report is the contained set of Multicast Address Records, which is illustrated in Figure 3.3. Each Multicast Address Record lists the multicast address and associated source addresses the listener is interested in.The Record Type field specifies whether the Report is in response to a Query, in which case the Report is a Current State Report, or if the Report is to inform the router that the listener has changed its multicast group interests, in which case it is a State Change Report. State Change Reports will be generated by listeners independently avoiding the need to wait for a Query from a router.

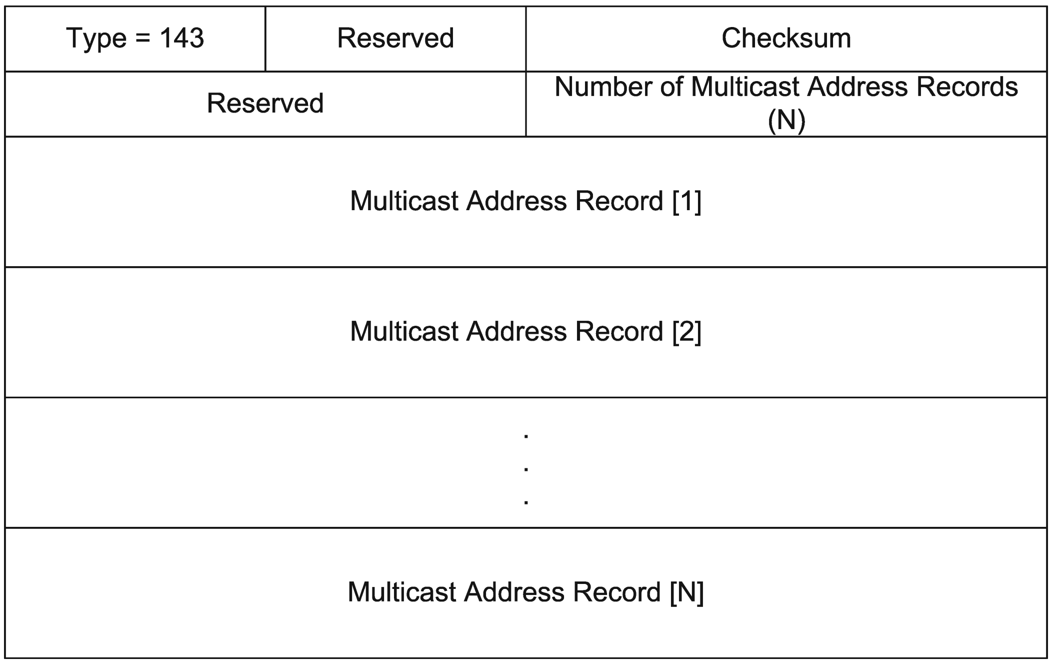

Type = 143
Reserved
Checksum
Reserved
Number of Multicast Address Records (N)
Multicast Address Record [1]
Multicast Address Record [2]
.
.
.
Multicast Address Record [N]

Figure 3.2: Multicast Listener Report Message [VC04]

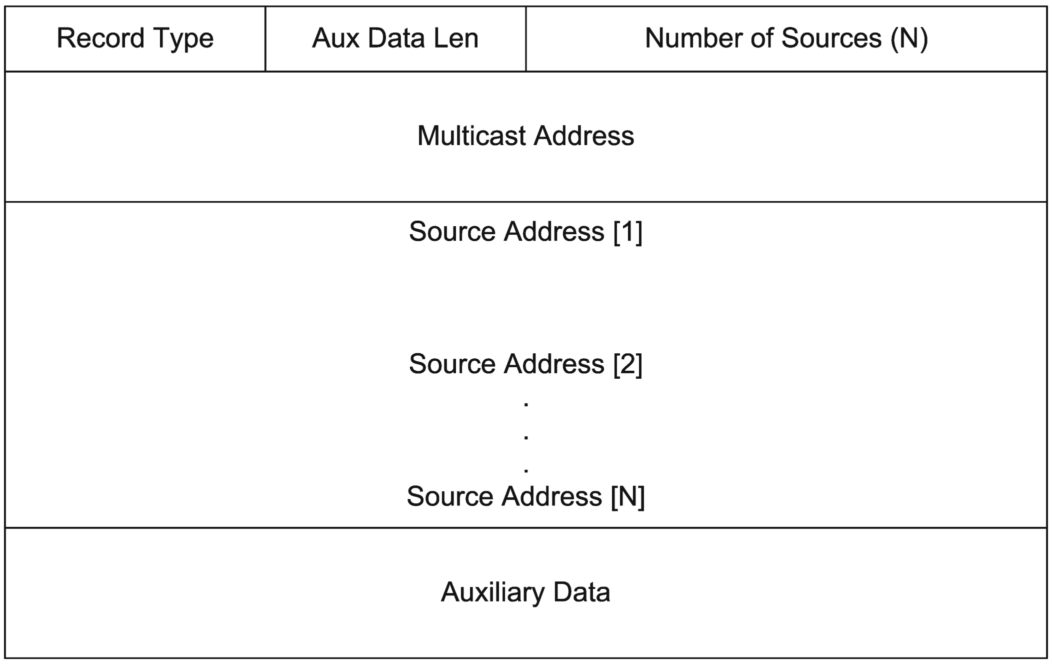


Figure 3.3: Multicast Address Record [VC04]

For example, if a listener wishes to subscribe to or unsubscribe from a multicast group it can send a State Change Report to the multicast router it is connected to. Upon receipt of this State Change Report, the multicast router, will either add or remove the listener's interests to or from its routing table.

If the multicast router receives a State Change Report which includes a multicast address that the router is not currently a multicast address listener for, then the router must send a subsequent State Change Report to all connected multicast routers, indicating this router is now a multicast address listener for that address. This will ensure that this router will receive data sent for that multicast address and can forward it on the relevant listener.

Similarly, if the result of the receipt of a State Change Report is that the multicast router no longer has any links interested in an address, the router must send a State Change Report to all neighbouring routers indicating it no longer wishes to receive multicast traffic associated with that address.

Through the exchange of Current State Reports and State Change Reports, multicast routers can maintain a knowledge of the interests of all multicast address listeners (clients and other routers) linked to the multicast router and thus deliver any packets it receives to the subscribed multicast address listeners.

The MLDv2 protocol allows source specific multicast to be implemented. Using source specific multicast, a multicast listener can specify a particular source that it wishes to receive or not receive multicast traffic from on the multicast addresses it is subscribed to. This is done using the Source Address fields in the Multicast Address Record.

3.1.1 Building Efficient Multicast Trees

Figure 3.4 illustrates how a multicast tree can be built using MLD Listener Reports. As clients and subsequent routers subscribe to a multicast address the multicast tree can become unwieldy and inefficient, as can be seen in the resulting multicast traffic illustrated in Figure 3.5. The tree constructed in Figure 3.4 would result in the

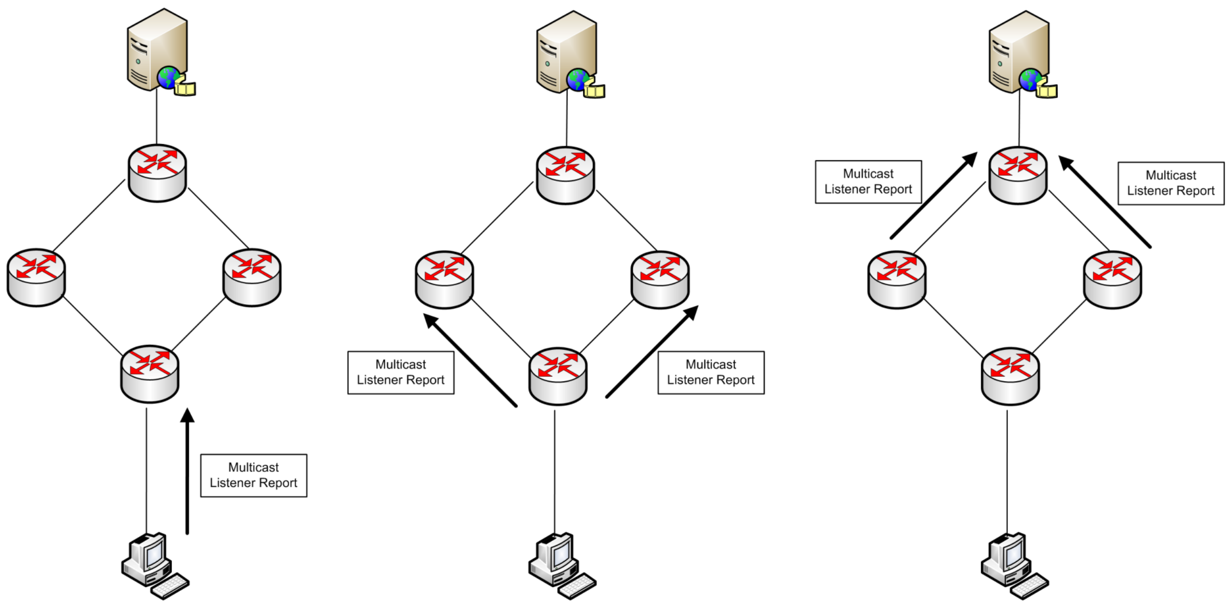

Multicast Listener Report
Multicast Listener Report
Multicast Listener Report
Multicast Listener Report
Multicast Listener Report

Figure 3.4: Multicast Subscription using MLD Listener Reports

unnecessary duplication of packets directed from the source to the client.

While the MLD protocol allows clients and routers to communicate their interest in specific multicast groups, it is insufficient by itself, for constructing efficient multicast trees and a further protocol is required. As the on-demand multimedia streaming architecture proposed in this thesis is based on a one-to-many service model with a centralised server (or servers) at the centre of the network, a protocol for implementing source specific multicast is appropriate.

The Protocol-Independent Multicast, Source-Specific Multicast (PIM-SSM) protocol [Bha03][PD07] is one such protocol. Using this protocol, clients specify which source (in this case the central server) they wish to receive multicast traffic from in their Multicast Listener Reports. PIM-SSM results in the construction of an efficient multicast tree from the source to the subscribed clients, by routing a single Multicast Listener Report towards the specified multicast source, instead of a Multicast Listener Report being sent to all connected multicast routers. In the case of on-demand multimedia streaming, the Report is routed from the client to the central server, thus resulting in an efficient multicast path from the server to the client. The building of a multicast tree using PIM-SSM and the resulting multicast traffic is illustrated in Figures 3.6 and

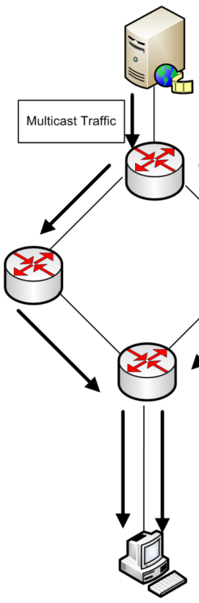


Figure 3.5: Multicast Traffic Resulting From an Inefficient Multicast Tree

3.7.

3.2 Multicast Multimedia Streaming

Due to the potential for reducing the bandwidth required for data delivery, multicast is an excellent option for high bandwidth applications where potentially large number of clients will receive the same data, as may be the case with multimedia streaming. Multicast is ideally suited for live streaming, where each subscribed client will receive the same content at the same time and does not expect individual control of the stream. This thesis explores the more challenging problem of using multicast to achieve efficiencies within on-demand multimedia services.

3.2.1 On-Demand Multicast Multimedia Streaming

On-demand multimedia streaming services can be classified as being *near on-demand* or *true on-demand* services. Near on-demand services are characterised by an intentional service produced latency in the response to client interaction. For example, there may be a significant delay between a client requesting playback of content, and the client

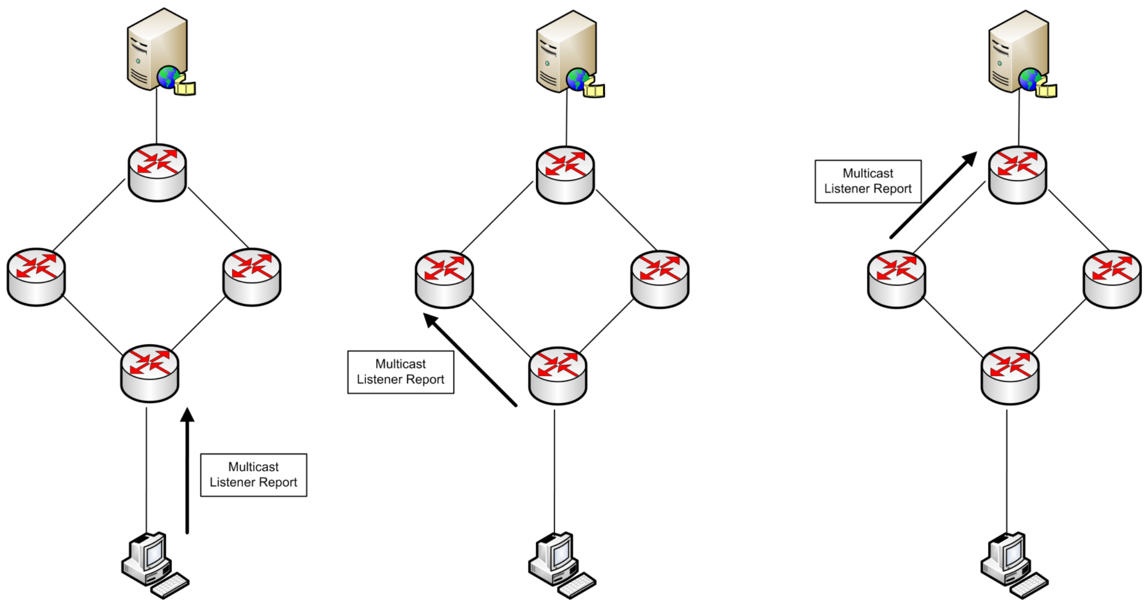


Figure 3.6: Multicast Subscription using MLD Listener Reports and PIM-SSM

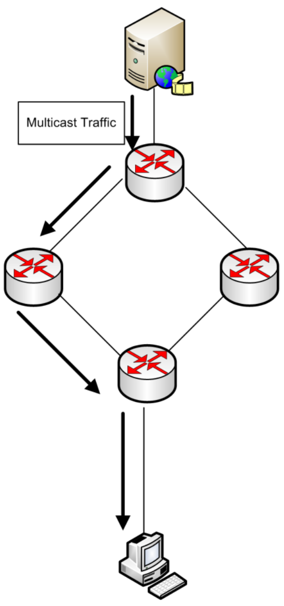


Figure 3.7: Multicast Traffic Resulting from PIM-SSM

receiving the start of the stream. With true on-demand services, in contrast, the response to a client request will be issued by the service as soon as possible.

A number of solutions have been proposed in the past for delivering on-demand multimedia streaming using multicast. For example, one approach proposed by Fei et al. [FAKM05] relies on clients buffering an incoming multicast stream and using the buffered content to implement interactivity for the client. Other approaches, such as those proposed by Liao et al. [LL97] and Fonseca et al [dFR05] used contingency channels in addition to the multicast channel to provide interactivity. Poon et al.[PLF05] proposed a combination of the above techniques to provide interactivity. Techniques such as Batching [And93] and Patching[HCS98] are based around exclusive use of multicast streams to deliver an on-demand service and these techniques are now described in detail.

Batching [And93] uses multicast to provide Near on-demand services. The basic premise of batching is to delay servicing playback requests for content items for a certain amount of time. This delay allows for requests for the same content items to be accumulated and when playback begins, all of the accumulated requests for the content are served with a single multicast stream. The key trade off with batching is balancing the number of requests served using a single multicast stream with the delay experienced by the users of the system. Scheduling policies for Batching, such as those proposed by Dan et al [DSS94] and Aggarwal et al. [AWY96] try to minimize both the average waiting time of clients and the likelihood of a client giving up on a requested stream while also ensuring that all requests are equally likely to be serviced. Batching techniques were originally proposed as an approach for handling requests to a fully utilised server, and these scheduling policies were used to select which multicast stream would be served when resources become available. In a modern context where service providers typically over provision for expected usage, the relevance of Batching is somewhat diminished.

Patching was proposed by Hua et al. [HCS98] as a way of using multicast to reduce network and server resource requirements, while maintaining clients' independent control of streams, thus offering a true on-demand service.

Using the Patching technique, a request from a new client will be serviced, where possible, by an existing multicast stream of the same content, referred to as a "batch" stream. Since the new client will have missed the beginning of the stream, it must begin buffering the batch stream while the server delivers the beginning of the content using a second stream. This second stream is referred to as the "patch" stream. Once the patch stream has caught up with the buffered content from the batch stream, the client can begin retrieving the remainder of the stream from the buffer.

Client buffer capacity is assumed to be a finite resource. If the missed portion of the stream is smaller than the buffer, then the patch stream only needs to provide this portion of the stream (Figure 3.8(a)). The original Patching technique proposed two different approaches to handle the case where the buffer capacity is insufficient to store the batch stream while the patch stream is being received. The first approach, called *Greedy Patching*, uses the patch stream to deliver the entire content stream, excluding the final portion of the end of the stream that can be buffered from the original batch stream (Figure 3.8(b)). In the second approach, called *Grace Patching*, the patch stream becomes a new batch stream and serves the entirety of the multimedia content (Figure 3.8(c)). It is important to note that both the patch and batch streams are delivered as multicast streams.

3.2.2 Implementing Patching Using RTSP

The Peer Assisted Multicast Streaming architecture proposed in this thesis shares some of the features of Patching. In particular, when a client requests a content item, it will receive the content in either one or two streams, which may be multicast streams. As a result of this similarity, the implementation of Patching is considered in detail as this was not addressed in the original work on Patching. A new approach to the implementation of patching using a real world protocol, in this case the Real Time Streaming Protocol [SRL98] (RTSP), is described in [OD09b].

An RTSP session can be viewed as having three phases. The first phase is the

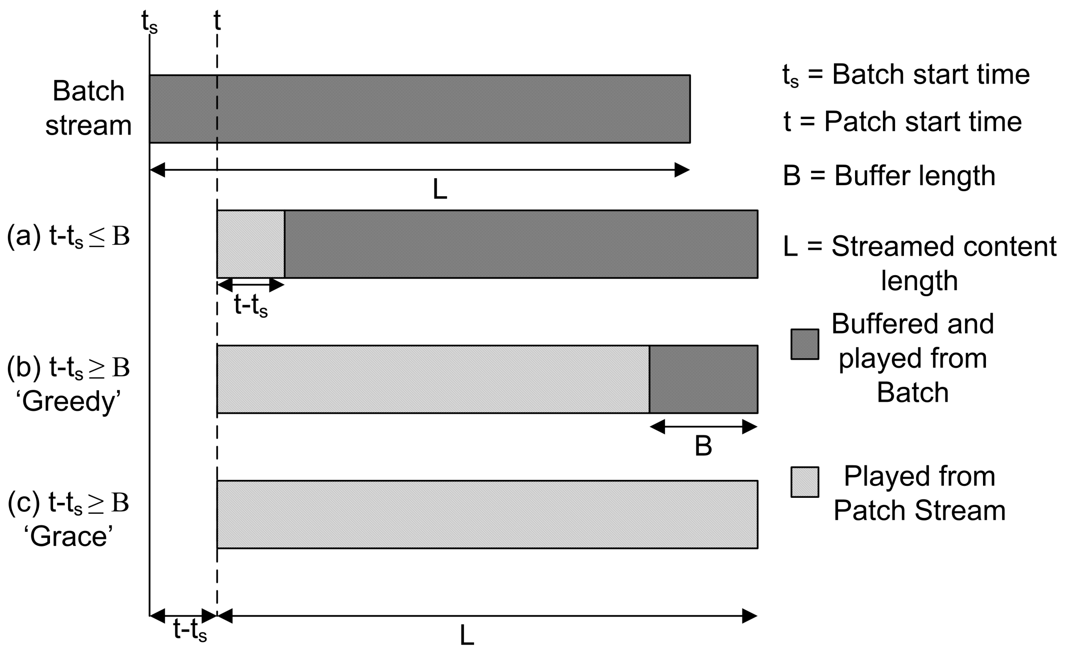


Figure 3.8: Timing and method of delivery of streams with Patching

negotiation between the server and the client. In this phase, which begins when the client issues an RTSP `SETUP` request, both parties must agree on all aspects of the transport mechanism that is to be used. Once the transport mechanism has been agreed, the multimedia stream can be delivered in the second phase, which begins when the client issues the first `PLAY` request. During this second phase, users can perform familiar stream control operations, such as play, pause and skip to, by issuing the relevant RTSP requests such as `PLAY` and `PAUSE`. The third phase is the tear-down phase, which is initiated when either the client or the server issues an RTSP `TEARDOWN` request to end the streaming session. A simple RTSP session is illustrated in Figure 3.9.

When using Patching to deliver media content to clients, there is a change in the relationship between the streaming control protocol (RTSP) and the associated stream (or streams) transmitted by the server. When a client exercises interactive control over an on-demand stream, by issuing RTSP `PLAY`, `PAUSE` or `TEARDOWN` requests, rather than directly causing the server to begin or end transmission of a transport stream, these requests will result in Patching policy decisions on the server. For example, a `PLAY`

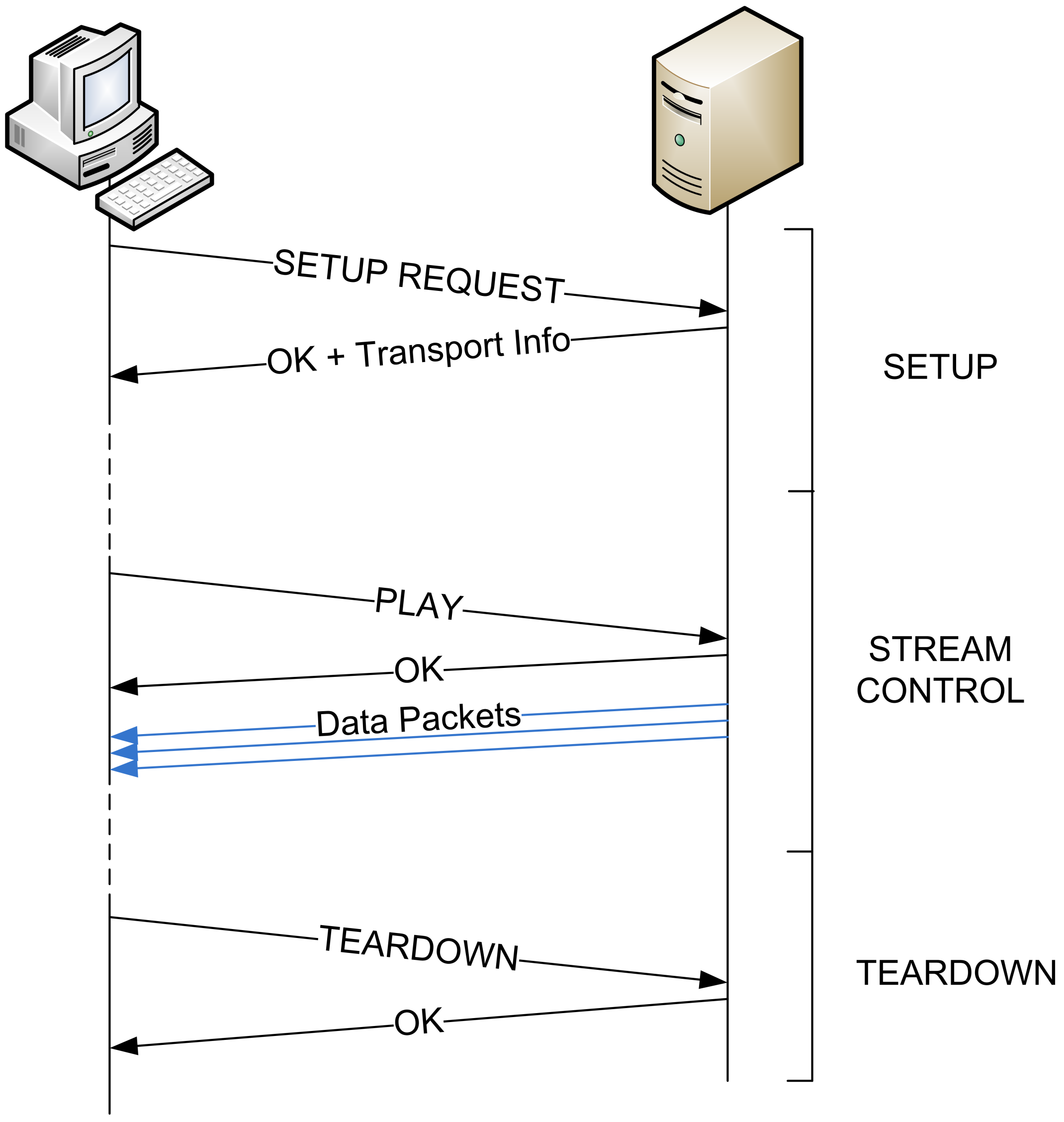


Figure 3.9: Simple RTSP Session

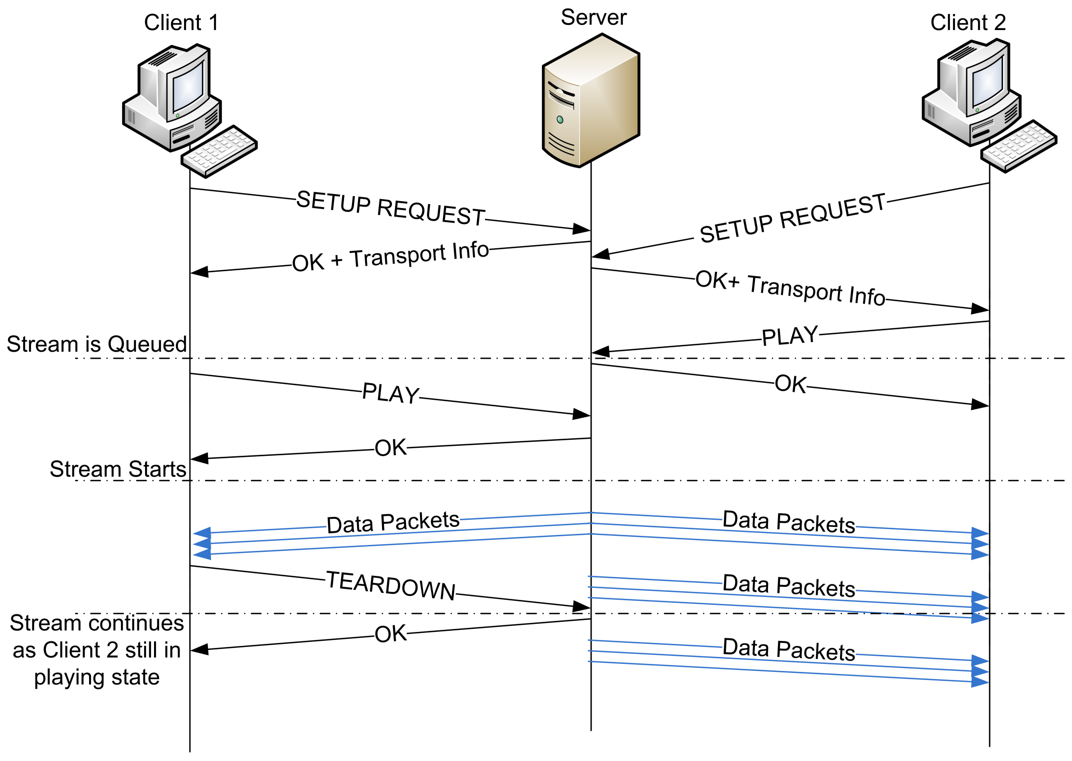


Figure 3.10: Patching RTSP Session

request might result in a client being assigned to a batch stream and a new patch stream, whereas a `PAUSE` or `TEARDOWN` request may have no effect on the multicast streams being transmitted by the server, unless the requesting client is the only client with an interest in the corresponding streams.

This relationship between streaming control protocol and the content streams transmitted by the server may be contrasted with the typical relationship that exists within a simpler unicast multimedia server. In the unicast server, on receipt of a `PLAY` request, the server will typically begin transmitting the content stream to the client and will stop transmitting the stream on receipt of a `PAUSE` request.

The transport information received in response to a client's `SETUP` request will contain the multicast address of the batch stream and, if applicable, the multicast address of the patch stream. The client must subscribe to these addresses to receive the requested media content. To avoid unnecessary traffic being routed to the client, the client will only subscribe to the addresses when it issues a `PLAY` request. To stop receiving a stream, a client will unsubscribe from the corresponding multicast address (or addresses) when it issues the `TEARDOWN` request. A more detailed description of the policy used by clients to determine when they subscribe to and unsubscribe from multicast addresses is detailed in section 3.2.3.

The transport mechanism, which according to the RTSP protocol must be agreed in the first phase of an RTSP session, may be invalidated in the second phase when the client interacts with the stream. Specifically, the transport mechanism, which includes the multicast addresses that the client should use to receive the multimedia stream, is established in the first phase. The server, however, can only make Patching policy decisions, specifically the assignment of clients to multicast groups, during the second phase when clients request operations such as `PLAY` and `PAUSE`. There will often be a delay between the client establishing the session with a `SETUP` request, and entering the second phase by requesting playback with a `PLAY` request. As a result, the initial assignment by the server of the client to a multicast stream in the first phase may be invalidated if the server has already commenced transmitting data to the multicast stream. This issue can be resolved within the bounds of the RTSP standard by instead of issuing an RTSP `OK` response to the `PLAY` request, the server issues an error response indicating that the transport information detailed in the original `SETUP` response is no longer valid and it will be necessary for the client to re-submit the original request. Once the response to this new `SETUP` request has been received, including new transport information and specifically a new multicast address, the client can immediately re-send its `PLAY` request, which on this occasion is likely to succeed. This behaviour is illustrated in Figure 3.11.

3.2.3 Multicast Subscription Policy

As previously stated the original presentation of Patching by Hua et al. did not address aspects of the implementation of patching using real world network protocols. In particular it is important to consider when Patching clients subscribe to and unsubscribe

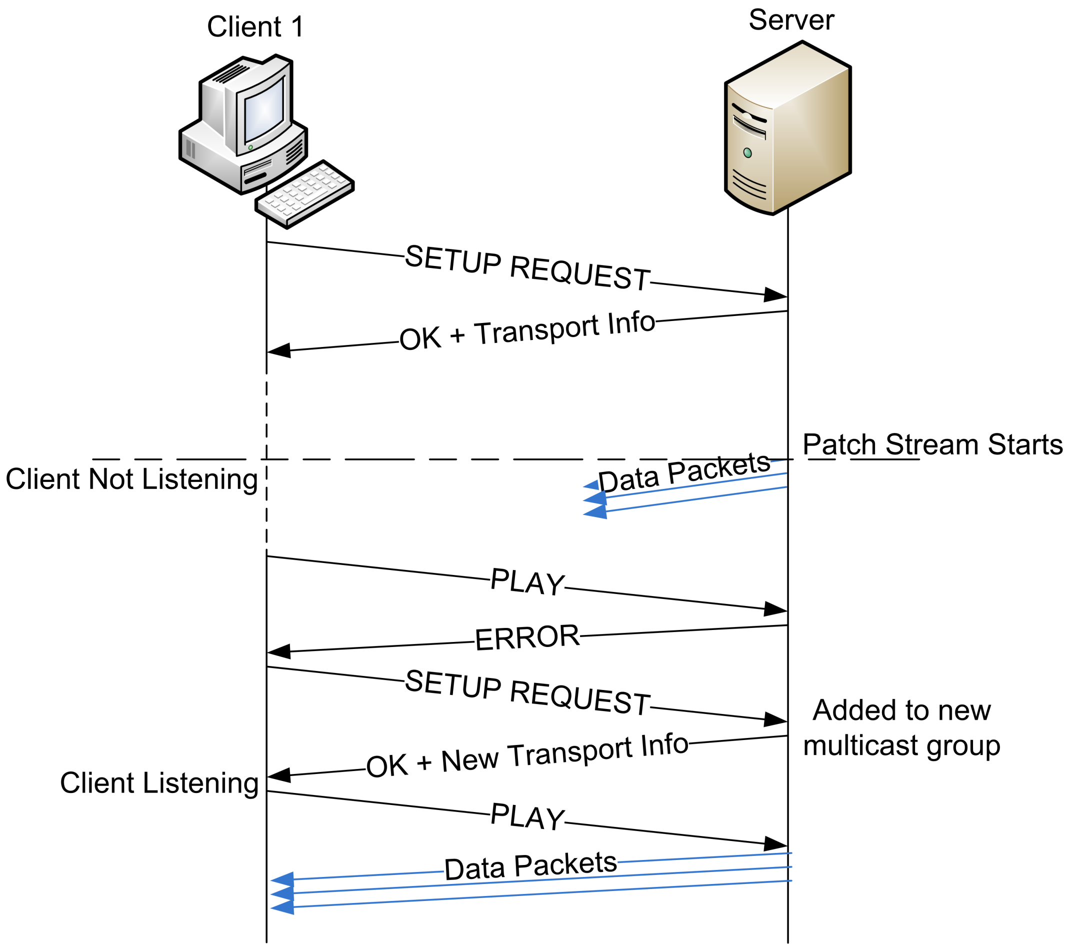


Figure 3.11: RTSP Patching session where the client "misses" the start of the stream

from batch and patch streams, as this will have an effect on network resource utilisation. As the Peer Assisted Multicast Streaming architecture also relies on multicast this is an important consideration.

The original work on Patching focused on server resources and server behaviour, treating everything external to the server as a "black box". To fully evaluate end-to-end network resource utilisation, however, it is necessary to model the behaviour of individual clients with respect to multicast stream subscription. As noted earlier, Patching requires each client to subscribe to either one or two multicast channels. If the server only returns a single multicast address to the client, that channel will contain the whole stream. If the server returns two channels then one of the following cases will apply.

Case 1 The missing portion of the Batch stream is smaller than the client's buffer. The beginning of the stream must be obtained from the patch stream, with the remainder obtained from the buffered batch stream. (Figure 3.8 case (a))

Case 2 The missing portion of the Batch stream is larger than the client's buffer and the server is using the Greedy Patching scheme. The first $L - B$ seconds of the stream will be obtained from the patch stream and the remaining B seconds are obtained from the buffered batch stream. (Figure 3.8 case (b))

Case 3 The missing portion of the Batch stream is larger than the client's buffer and the server is using a Grace Patching scheme. The entire stream must be obtained from the patch stream. (Figure 3.8 case (c))

Case 4 The batch stream has ended by the time the patch stream starts. The patch stream will serve the entire stream regardless of whether the server is using a Greedy or Grace Patching scheme. This case arises as a result of the implementation of Patching using RTSP, as the associated batch is chosen when the client issues a `SETUP` request. In the original Patching paper the associated batch was chosen when the Patch stream was to be served so this case would never arise.

Two approaches are proposed here to determine when clients subscribe to and unsubscribe from multicast patch and batch streams. These approaches are termed *full subscription* and *minimum subscription.*

In the *full subscription* policy, when the server returns two multicast group addresses to a client, the client subscribes to both multicast groups for the duration of the streaming session. While this scheme lends itself to a simple implementation, it also results in the receipt of substantial quantities of duplicated stream data. For example, take a buffer length of 5 minutes, and a stream length of 90 minutes. In *case 2*, if the patch stream started half way through the batch stream, then the client would receive a patch for 85 minutes, and buffer the last 5 minutes of the batch. But it would also receive 40 minutes of redundant data from the batch stream. This means that for 44% of the stream duration, the client is using double the bandwidth of a single stream, despite half of this data being redundant.

Similarly, in case 3, the patch stream would provide the entire 90 minutes of the multicast stream, and the original batch would provide 45 minutes of redundant data. This means that for the first half of the session the client is using up to twice as much bandwidth as is necessary.

It is sufficient when conducting a server-centric evaluation to simply model *full subscription*, since the server will transmit data for every multicast stream with at least one subscriber. However, to observe the real benefit of multicast over an entire network, it is necessary to model a more efficient multicast subscription policy, which is termed *minimum subscription.*

With *minimum subscription*, the client only subscribes to each multicast stream when required in order to receive the entire content stream. The receipt of unnecessary duplicated data can be avoided by unsubscribing from the Batch stream when possible.

To implement *minimum subscription*, the client must still subscribe to both streams initially. On receipt of the first packets of the patch and batch streams, the data received can be examined and the client can decide which of the above cases applies.

If case 2 applies, then the client can analyse the batch stream data and calculate how long it will be until the batch stream will provide useful data. The client can then unsubscribe from the batch stream and resubscribe at a later time. This can be only be done if the server is sending the multicast data packets at a fixed rate. To compensate for jitter in the network, the client may choose to subscribe slightly earlier than the calculated time, to avoid data loss. A client may estimate how much earlier it should resubscribe to the multicast stream based on an estimation of network performance.

In case 3, the client can unsubscribe from the batch stream and will not need to resubscribe, as the patch stream will provide the entire content stream.

These unsubscriptions are independent of the RTSP control protocol, as it is not necessary for the server to be aware when the client has unsubscribed from or resubscribed to the batch stream.

3.2.4 Evaluation of Multicast Patching

A performance evaluation of multicast Patching was performed using the network simulation framework discussed in Section 6.4. This evaluation differed from the original evaluation of Patching which took a server-centric approach to performance evaluation. The original evaluation modelled the external network and clients as a "black box", whereas the new evaluation took into account the impact of a realistic network topology and real network protocols.

The results of this evaluation showed, in a similar manner to the original evaluation of Patching, the benefit of the Grace Patching policy over Greedy Patching. The results differed though in showing that the behaviour of both the client and server will have a significant effect on performance, and that this effect is closely related to network distance from the origin server. This effect is shown in Figure 3.12.

The server's choice of Patching scheme has its greatest effect closer to the server. The clients' behaviour can alter the network bandwidth utilisation both at the edges of the network and at the server. In particular, the effect of client multicast subscription behaviour is greatest at the edges of the network while increasing the size of a client's

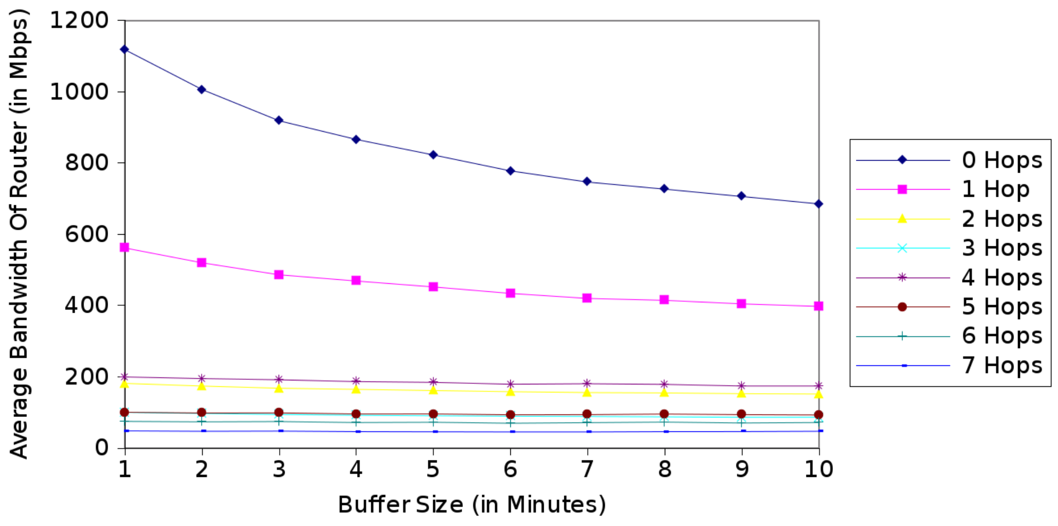


Figure 3.12: Effect of Buffer Size on Grace Patching Using Minimum Subscription at Increasing Network "Hops" from the Server

buffer has a diminishing effect as the network distance from the server increases.

3.3 Summary

This chapter has described how multicast may be used to improve the performance of on-demand multimedia streaming applications. Maintaining fully interactive client control, however, is a significant challenge, and a number of solutions have been previously proposed. The Peer Assisted Multicast Streaming (PAMS) approach proposed in this thesis is inspired by the prior work on Patching, but differs in the way that interactive is maintained when using multicast. Specifically, PAMS uses peer-to-peer technology to provide interactivity while receiving multicast streams from a server that effectively implements Batching. The PAMS architecture is described in detail in Chapter 5 while the peer-to-peer technology upon which it is based is described in the next chapter.

The original works on Batching and Patching did not fully address some of the real-world challenges of implementing these policies and solutions to these challenges which may be applied to the implementation of PAMS have also been described in this chapter.

Chapter 4

Peer-to-Peer Multimedia Streaming

Peer-to-peer networking is a distributed architecture, in which the workload and responsibilities of the application are usually distributed among the participating peers. In a peer-to-peer application, peers both supply and consume resources within the network. This differs from the general client-server model in which the server is the supplier and the client is the consumer of a service.

Peer-to-peer applications are scalable due to their decentralized approach. Examples of peer-to-peer applications include BitTorrent for file-sharing [Coh03], Skype for Voice over IP [1], and SOPCast [2] for live multimedia streaming. Peer-to-peer applications attempt to balance the load fairly amongst all peers participating in the application.

One of the challenges that must be addressed by a peer-to-peer application is the continual arrival and departure of peers. This is referred to as churn [RGRK04] and may significantly impact the performance of the application. The impact of churn have been highlighted and solutions proposed in the literature [RGRK04].

Peer-to-peer applications are usually built on top of peer-to-peer architectures

[1] `http://www.skpe.com`
[2] `http://www.sopcast.com`

which are independent of the application. The underlying architectures are generally tasked with locating peers or data within the network, and for maintaining the structure of the peer-to-peer network. Peer-to-peer architectures are usually implemented as Overlay Networks in which the Overlay Network structure is implemented on top of an existing network structure. One such Overlay Network is Chord [SMK+01] and a description of this architecture will now be presented.

4.1 Chord Overlay Network

Overlay networks such as Chord [SMK+01], Pastry [RD01] and Tapestry [ZHS+04], among others, provide routing mechanisms that facilitate finding peers or data within the overlay network. Within a peer-to-peer network it may not be possible for a peer to route a message directly to another peer without knowing the exact location of the peer or data in the underlying network.

Within a peer-to-peer network, while a peer may know the identity of another peer within the context of the overlay network, it may not know the underlying network identity of that peer. As a result the peer cannot directly send a message to the other peer. The overlay network provides a mechanism to route messages between any two peers, by ensuring that each peer maintains knowledge of the underlying network identities of a subset of the peers.

For the general cases of Pastry and Chord, each peer within the overlay network is given an identifier usually by applying a hashing function. Furthermore, data items may be assigned identifiers within the overlay network, allowing data items to be associated with peers. For peers, often the peer's IP address is used as the input for the hashing function, while a data item's name can be used to derive the item's identifier from the hashing function. In Chord the node whose identifier is the closest subsequent identifier to the item's identifier is the node responsible for the data item. Alternatively, in Pastry, the node with an identifier numerically closest to an item's identifier is responsible for knowledge of that item.

4.1.1 Chord Routing

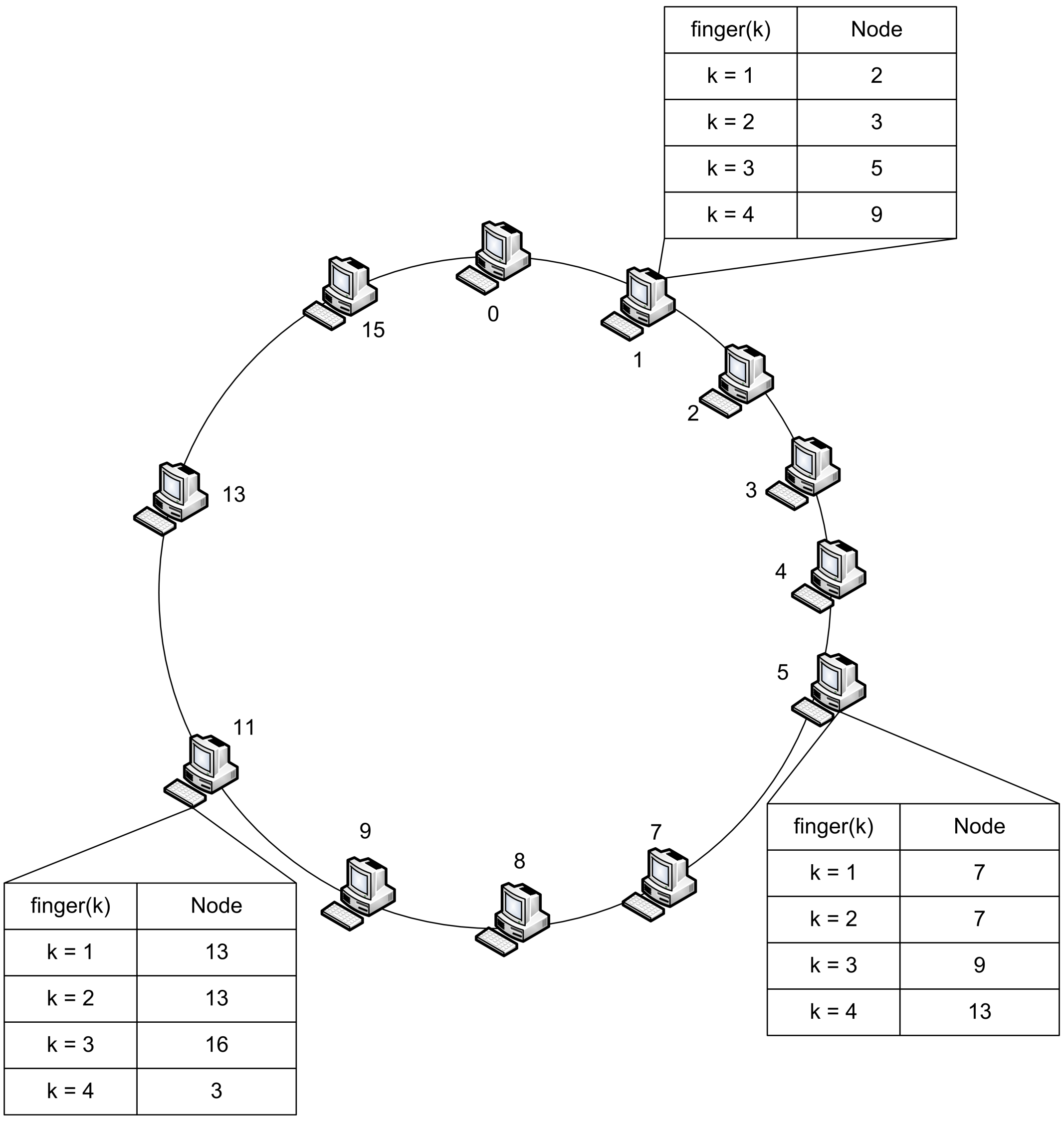
Each node within the overlay network need only know about a small subset of the other nodes in order to route messages to any destination. If a Chord node receives a message to be routed to a key higher than it's own, it need only forward the message on to a node that has a higher identifier than itself. If each node was only aware of its immediate successor, than routing could be achieved but would be inefficient and the overlay network topology would be a ring.

To improve on this, each node with identifier n maintains a small routing table, of maximum size m, where m is the size of the identifiers in bits. Each i^{th} entry in the table corresponds to a node identifier that is greater than the current node identifier by at least 2^{i-1}. The routing tables are circular, so the identifiers are modulo 2^m. Each of these successive nodes are referred to as fingers and the nearest finger (i=1) is referred to as the *successor*. An organisation of the routing tables for a Chord network with a identifier size of m=4 is shown in Figure 4.1. The structure of these tables is such that nodes will know a lot about numerically close nodes, and less about nodes which are numerically distant.

The tables illustrated are further simplified for missing fingers, so that the range of fingers for each entry is recorded, for example, for Node 11 in the illustration in Figure 4.1, the routing table would need only one entry to represent the first and second fingers.

Routing is performed by a node by sending the message to the finger that is the nearest predecessor, that the node is aware of, to the destination identifier. An example of Chord routing is shown in Figure 4.2. In this example, Node 1, is attempting to route a message to Node 8. The nearest predecessor that Node 1 is aware of is Node 5, so the message is routed here. From here the message is routed to Node 7, and from Node 7 the message is finally delivered to Node 8.

Maintaining these routing tables is critical to the reliability of an overlay networks, but the tables do not need to be fully correct in order to guarantee that a message can be routed to a destination. Upon a node joining or leaving, the tables

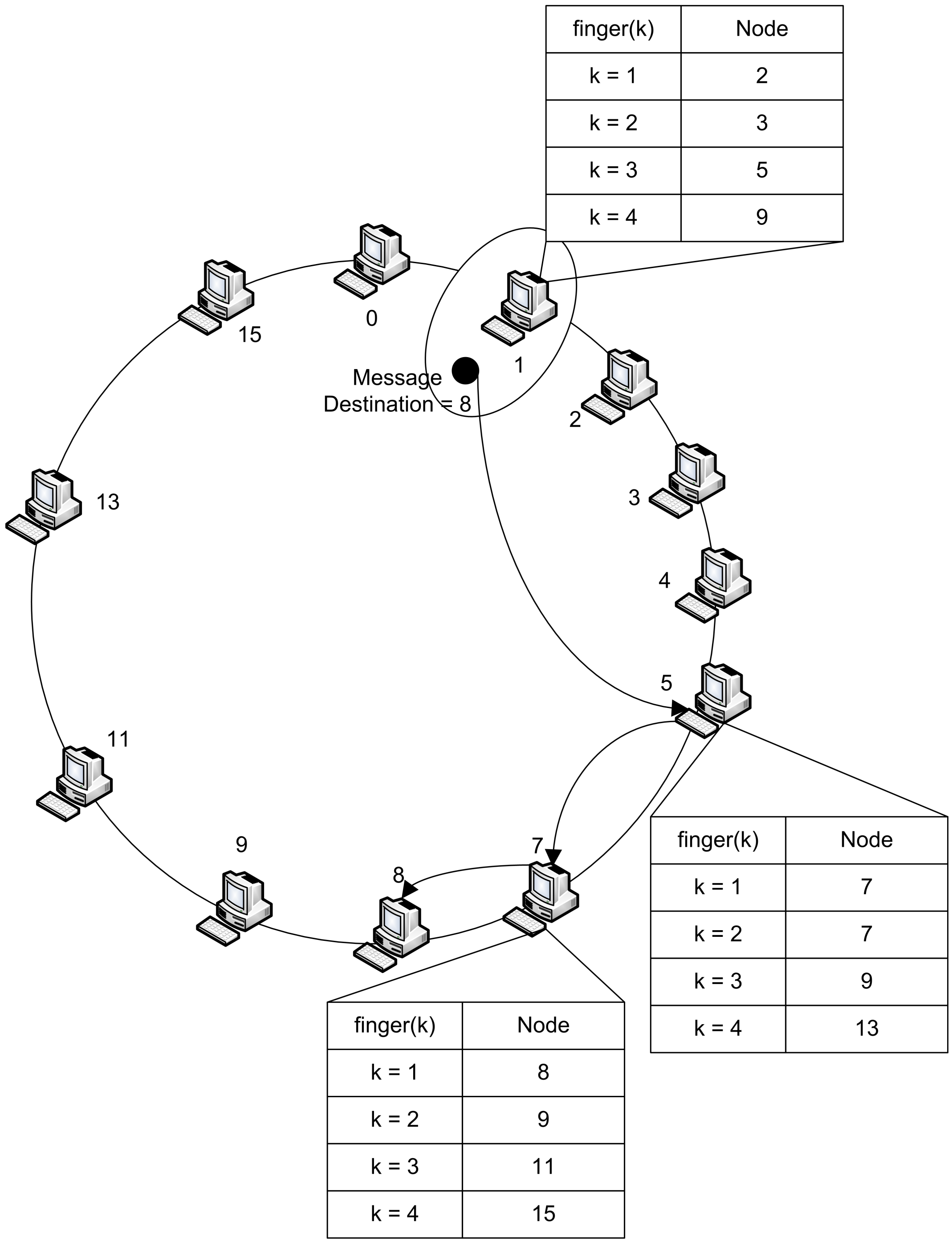


finger(k)	Node
k = 1	2
k = 2	3
k = 3	5
k = 4	9

finger(k)	Node
k = 1	7
k = 2	7
k = 3	9
k = 4	13

finger(k)	Node
k = 1	13
k = 2	13
k = 3	16
k = 4	3

Figure 4.1: Chord Routing Tables



finger(k)	Node
k = 1	2
k = 2	3
k = 3	5
k = 4	9

finger(k)	Node
k = 1	7
k = 2	7
k = 3	9
k = 4	13

finger(k)	Node
k = 1	8
k = 2	9
k = 3	11
k = 4	15

Figure 4.2: Chord Message Routing

must be updated. This can be done directly when a node joins and through a periodic stabilization procedure which is omitted here but is detailed in original work on Chord [SMK$^+$01].

4.1.2 Application Example: Squirrel

Applications can be designed and built on top of overlay networks. One such application is Squirrel [IRD02], which provides a decentralised web cache built on top of Pastry, and this will now be described in detail.

Pastry [RD01] is similar in design to Chord, except for a few key differences. In Chord, a message was directed to the nearest predecessor of the destination, while in Pastry the message is directed towards the node that is nearest numerically to the destination. This requires routing tables to know more about their immediate predecessors as well as their successors, but routing is still achieved in a similar fashion, with multiple hops required to successfully route a message.

Squirrel was designed as a way of reducing the incoming bandwidth requirements of a network caused by accesses to web content without requiring additional hardware to implement a large cache for all requests. Squirrel uses each individual peer's local web cache to provide resources to a large decentralised collaborative web cache to serve content to the other peers.

As with all peer-to-peer applications, the main feature of Squirrel is the manner in which content is located within the overlay network. Each peer is assigned an identifier using a hashing function, such as SHA-1 [FIP95]. To locate cached content for a specific web address, or URL, the URL is also hashed, giving an identifier for the content. A lookup request is now routed to the peer whose identifier is numerically closest to the content's identifier. This peer is called the *Home Node* for that content. How the *Home Node* delivers the content (if cached) to the peer depends on the Squirrel scheme used, of which there are two, the *Home-store* scheme or the *Directory* scheme.

In the *Home-store* scheme, the *Home Node* is responsible for storing the content and delivering it to the requesting peer. If the Home Node does not have a copy of the content it must retrieve it from the server and forward it to the requesting peer.

In the alternative *Directory* scheme, any node which has stored the content, informs the *Home Node* that they have a copy of the content. These peers are referred to as *Delegates*, and the *Home Node* stores a list of *Delegates*. Upon receiving a request for content from a peer, the *Home Node* forwards the request to a *Delegate* and the *Delegate* provides the content to the requesting peer.

Evaluations of Squirrel showed that, with a small cache at each peer, a Squirrel web cache could provide a significant reduction in the external network traffic produced by web accesses. The *Home-store* scheme performed better than the Directory scheme, but the *Home Node* scheme can produce a higher load on individual peers if they are tasked with supplying the content of frequently accessed pages.

The Peer Assisted Multicast Streaming architecture proposed in this thesis, builds on the ideas proposed for the Squirrel decentralized web cache.

4.2 Peer-to-Peer content delivery

In recent years, with the growing popularity of peer-to-peer systems such as BitTorrent [Coh03] and KaZaa [LRW03], peer-to-peer streaming services have also been developed. Examples include SplitStream [CDK+03] for on-demand streaming and CoolStreaming [ZLLsPY05] for live media streaming. Both of these systems are based on the use of overlay networks. They do not rely on central servers that contain all the content and therefore there is less likely to be a bottleneck at any point in the network. The clients themselves host the content and distribute it to other clients. These systems claim to be extremely scalable as the more popular they become, the more resources are available to serve streams.

Implementations vary in the methods used to store and distribute content. SplitStream encodes one media stream into a number of independent "stripes". Each one of these stripes is distributed among numerous clients. When a client wishes to view the stream, it receives the stripes of the stream from a number of clients and these stripes are recombined to produce the requested stream. This technique is referred to as Multiple Description Coding (MDC). This approach differs from approaches similar to that proposed by Xiang et al. [XZZ$^+$04], in which one client is responsible for storing the entire media stream and for serving it to a user.

In both of these systems, the replication strategies used for content placement are important. If a popular file is not replicated enough throughout an overlay network then the clients hosting the file will be overloaded with requests. Similarly, the network location of the host chosen to supply a stream will influence latency.

Another issue with peer-to-peer streaming is that it is difficult to guarantee availability. This is especially an issue with unpopular content as most replication strategies will place unpopular content on a small number of nodes. If these nodes become unavailable then there is a chance the content could be lost or unavailable for a period of time.

Fairness is another concern within peer-to-peer systems. Clients should contribute to the resource needs of the system in a fair and balanced manner. In poorly balanced systems single clients may be required to provide large amounts of content while other clients provide very little resources to a system. The issue of fairness has been examined by Shrivastava and Banerjee [Shr05] among others.

4.2.1 Hybrid Architectures

Hybrid architectures have been proposed which combine a centralised server architecture with peer-to-peer techniques. An example of a hybrid architecture is CoopNet [PWCS02]. CoopNet has been developed for both live streaming and for on-demand streaming. For live streaming, the clients are used to reduce the bandwidth utilisation of the server by providing application level multicast. For on-demand streaming, the server provides a requesting client with information about content hosted on other peers. This information can be used by the requesting client to retrieve the content hosted on other peers. CoopNet is primarily focused on the live streaming aspect of the application though, whereas the on-demand streaming aspects have been left unresolved.

4.2.2 Application Level Multicast

Due to the limited deployment of multicast compatible infrastructure for network level multicast, peer-to-peer technologies known as Application Level Multicast (ALM) have been developed. ALM uses peer-to-peer networks to manage subscriptions to a multicast address and to route and deliver multicast traffic to interested parties. ALM is beneficial to users and content providers due to the scalability provided by being based on peer-to-peer technologies, but they are subject to reliability issues due to high churn within the participants of the ALM system [AGP12]. Some examples of ALM systems are Scribe [CDKR02] which is built upon Pastry and the ALM system designed by Ratnasamy et al [RHKS01].

4.3 Combining Peer-to-Peer and Multicast Streaming

The Peer Assisted Multicast Streaming architecture proposed in this thesis seeks to reduce the load on a centralized server by combining multicast and peer-to-peer technologies. A small number of other proposals exist which rely on similar principals, and these will now be described.

One architecture that combines multicast and peer-to-peer streaming is the Peer-to-Peer Batching Policy proposed by Ho et al. [HPL06b]. This architecture was later refined as the Enhanced Peer-to-Peer Batching Policy [HPL06a]. This architecture seeks to reduce server bandwidth requirements as follows. When a server receives a request for a content item, the server begins to serve this request with a multicast stream. Any subsequent requests that arrive within a specified time window are also served by the same multicast stream, with the missed portion being served by the client which last requested the content. In this way, the first client serves the second client, the second client serves the third client and this pattern continues for subsequent requests thus creating a chain of peers supplying the missed portion of the multicast stream. A similar scheme, called P2Prefix Patching was proposed by Lee et al. [LGJ$^+$06], and a similar application for use in an enterprise network was proposed by Farhad et al. in [FMAHK09].

These architectures, however, differ significantly from the Peer Assisted Multicast Streaming architecture proposed here. They differ in the way content is managed, located and used. All of these existing architectures use a central server to locate buffered content and select peers to service new client requests, in contrast to the decentralised approach proposed in this thesis. Furthermore, the collaborative prefix caching approach that is proposed here caches content for long periods across multiple streaming sessions, thereby allowing more extensive use of peer resources.

4.4 Summary

This chapter has described how individual peers can combine their resources to implement distributed applications, in which each peer both provides to and consumes from a service. Peer-to-peer based applications can be used as an alternative to centralized delivery for multimedia content. The manner in which the Peer Assisted Multicast Streaming architecture will use peer-to-peer technologies will now be described in detail in Chapters 5 and 6.

Chapter 5

Peer Assisted Multicast Streaming

In this chapter a new architecture for on-demand multimedia applications called Peer Assisted Multicast Streaming [OD09a], PAMS, is presented. The main benefit of the architecture is to reduce the server and network bandwidth required for delivering popular content. Peer Assisted Multicast Streaming combines multicast and peer-to-peer technologies. As has been discussed in Chapter 3, the server and network bandwidth required to deliver popular content can be reduced by serving multiple clients with a single multicast stream. This, however, prevents an individual client from controlling the stream they receive. This loss of interactivity can be overcome using techniques such as Patching, as discussed in Section 3.2.1. The patch streams that this technique relies on, however, will result in many short-lived streams, placing an additional load on the server.

The solution proposed here uses resources located at the edge of the network to deliver these short lived patch streams, removing their associated overhead from the server. In order to achieve this, upon receiving a stream, a client will use local storage to cache the prefix, or beginning, of the content. Clients then collaborate to provide each other with the prefixes that they have cached. Collaborating clients use a peer-to-peer overlay network to allow prefixes to be located within the collaborative cache.

This collaborative cache can then be used to provide the prefix of a content stream, while the remainder of the content will be provided to the client by the server using a multicast stream.

This chapter will be begin by exploring the rationale behind Peer Assisted Multicast Streaming and describe its behaviour. The focus of this chapter will be the design of the collaborative prefix cache, since the proposed architecture only requires basic support from a server.

5.1 Rationale

Both multicast systems and caching systems are most effective at reducing the resource costs of supplying the most popular content items in an on-demand streaming service. According to the locality principle, if a client requests a popular content stream, it is likely that another client will have requested the same content stream in the recent past. Multicast streaming approaches exploit this locality principle by using a single multicast stream to serve those clients whose streams are at similar playback positions, which is most likely to occur for the most popular content streams [And93]. Caching exploits the same locality principal by using resources at the edges of the network to supply streams of recently requested content, thereby reducing server load. The most popular content streams are the ones most likely to be available to be served from the cache.

The simplest way for a server to service multiple clients with a multicast stream is to delay the start of the stream, allowing time for subsequent requests for the same content to arrive. All of the requests arriving within this delay period can then be served by a single multicast stream. The challenge when using multicast in this way, however, is to retain the expected level of client interactivity and minimise latency while maximising the number of clients sharing a multicast stream.

Batching [And93], which has been described in Section 3.2.1, was one multicast streaming approach that was proposed for situations where the server did not have enough resources available to service all requests or in which the servicing of requests is intentionally delayed in order to reduce resource utilisation. Using Batching, requests are queued up, batched together, and served in a single multicast stream when there is sufficient outgoing bandwidth available to the server or when the delay has elapsed. As batching requires delaying streams to achieve sharing, this results in a high latency for the beginning of playback for most of the requesting clients.

Patching [HCS98], which has been described in Section 3.2.1 was proposed as an alternative multicast streaming approach, which allowed clients to share multicast streams without delaying playback. Patching exploits the ability of clients to buffer existing multicast streams, while receiving the missed prefix using a separate stream. When using Patching the server provides clients with *patch streams* containing these missing prefixes. Patch streams still contribute towards network congestion near the server and, as outlined in the original paper, need to be delivered as multicast streams, even if the stream is not shared at the time that it starts. The potentially high number of short lived multicast streams may make Patching unattractive in practice.

As has been discussed, multicast based approaches, such as Batching and Patching reduce the outgoing bandwidth utilisation by a server and relieve network congestion near the server, while still providing an on-demand streaming service to clients. This reduction in bandwidth requirements is achieved by reducing the number of concurrent streams required to service all requests for the most popular content items. When serving less popular content items, for which streams are not shared, these approaches will have no benefit over a simple unicast approach.

The PAMS architecture proposed here will further relieve network congestion near the server. This is achieved by using resources at the edges of the network to serve the patch streams containing the missed content prefix that are required by a Patching system.

To allow the prefix streams to be served using resources at the edge of the network, prefixes received by clients will be cached and made available for re-use by other nearby clients. There are a number of approaches that may be used to cache the content.

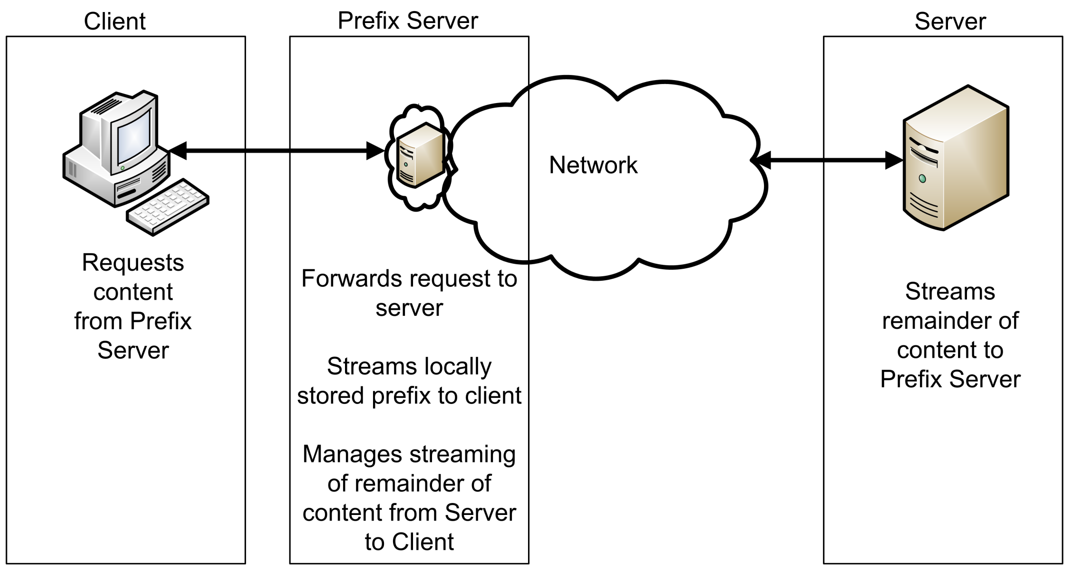

Client
Prefix Server
Server
Network
Requests content from Prefix Server
Forwards request to server
Streams locally stored prefix to client
Manages streaming of remainder of content from Server to Client
Streams remainder of content to Prefix Server

Figure 5.1: Proxy Prefix Caching

Firstly, a client could cache prefixes locally for later re-use but this approach would prevent other nearby clients from benefiting from the cached content and the likelihood of the prefix being re-used is low. Secondly, dedicated proxy servers may be deployed at the edges of the network to cache and serve prefixes to requesting clients. Alternatively, in the approach proposed here, clients may co-operate by combining the contents of their local caches to implement a distributed, collaborative cache and serve prefixes to each other.

An example of the second approach, in which proxy servers are deployed at the edges of the network, is the *Proxy Prefix Caching* architecture [SRT99], which was previously discussed in Section 2.4.3. When using a Proxy Prefix Cache, the content received by clients originates from two sources. The start of the content is served from a prefix cache, which is a server located close to the requesting client. The rest of the content is then streamed from the server through the prefix cache and onto the client. This results in a lower latency for stream startup, Figure 5.1 illustrates the roles of a proxy prefix caching system.

The main disadvantage of this second approach is the requirement for dedicated proxy servers to be distributed around the network, which may be costly to maintain

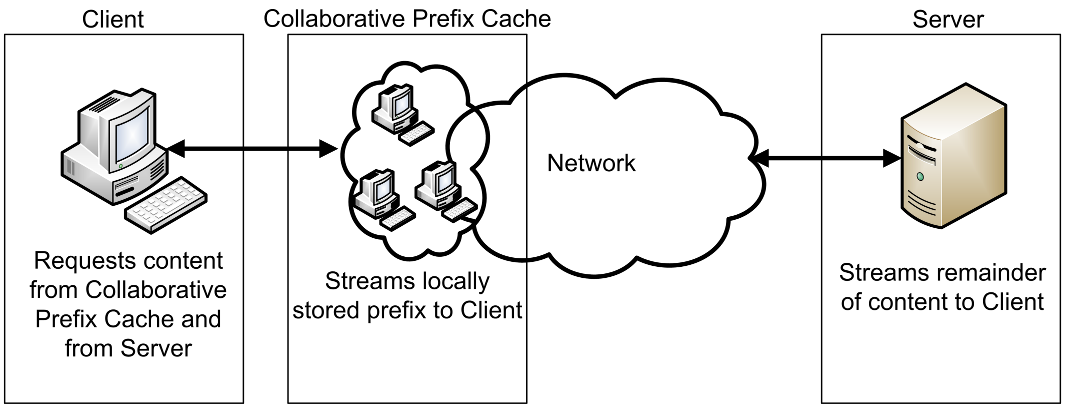

Client
Collaborative Prefix Cache
Server
Network
Requests content from Collaborative Prefix Cache and from Server
Streams locally stored prefix to Client
Streams remainder of content to Client

Figure 5.2: Collaborative Prefix Caching

and lack scalability. The original work on Proxy Prefix Caching did not investigate combining the approach with multicast.

In the alternative approach that is proposed here, clients co-operate in a collaborative cache by providing the prefixes stored in their local caches to other clients. This approach has the advantage of delivering a scalable solution, without requiring any additional dedicated infrastructure. A collaborative cache architecture is illustrated in Figure 5.2.

The collaborative caching approach builds upon the Squirrel [IRD02] architecture, which provided collaborative caching for web content. Like Squirrel, the proposed approach is implemented using a peer-to-peer overlay network. While web content delivery and multimedia streaming have inherently different properties, the advantages a collaborative cache can deliver are the same. The collaborative cache is designed to be entirely decentralised, so has no single point of failure. The decentralized architecture allows the workload associated with delivering prefixes to be distributed across the network, which will reduce the occurrence of choke points resulting in network congestion.

The nature of delivering multimedia content, in contrast with web content, however, requires the collaborative caching approach used by the Squirrel architecture to be adapted. Clients will have limited outgoing bandwidth, and as multimedia streaming is both resource intensive and long lived, the burden placed on clients participating in the collaborative cache is a concern unless limits are placed on the number of streams clients are expected to concurrently deliver. Also, while decentralized lookup avoids a single point of failure, it will introduce delays in the time taken to locate content, which will be reflected in the end user experience as a higher stream startup latency. Finally, there will be an overhead associated in managing content in the cache, and this must be kept to a minimum.

In the proposed Peer Assisted Multicast Streaming architecture, a collaborative prefix cache is used by the clients to reduce the load on a central multicast server. This is achieved by allowing the server to focus on serving long lived streams while the collaborative cache provides short prefixes of content items. By retrieving the prefix from the collaborative cache, clients will enable the server to delay serving the remainder of the content. This delay will offer the server an opportunity to service multiple requests with a single multicast stream, while still providing a true on-demand service.

As the architecture combines both a centralised approach with a decentralised one and as the two are independent of each other, the performance will degrade gracefully towards that of a centralized server as the performance of the collaborative cache deteriorates.

Peer-to-peer approaches, such as the proposed collaborative cache, are often considered as unattractive by content providers, due to concerns over Digital Rights Management (DRM), as it can be difficult to maintain control over the distribution of content. The approach proposed here, however, in which only the prefixes are cached and the rest of the content is still delivered from a centralised server, would allow providers to maintain control over part of their content while still benefiting from the reduction in bandwidth requirements that peer-to-peer systems can deliver in distributing content.

The operation of the proposed Peer Assisted Multicast Streaming architecture will be described in detail in the subsequent section, with a particular emphasis on the operation of the collaborative prefix cache.

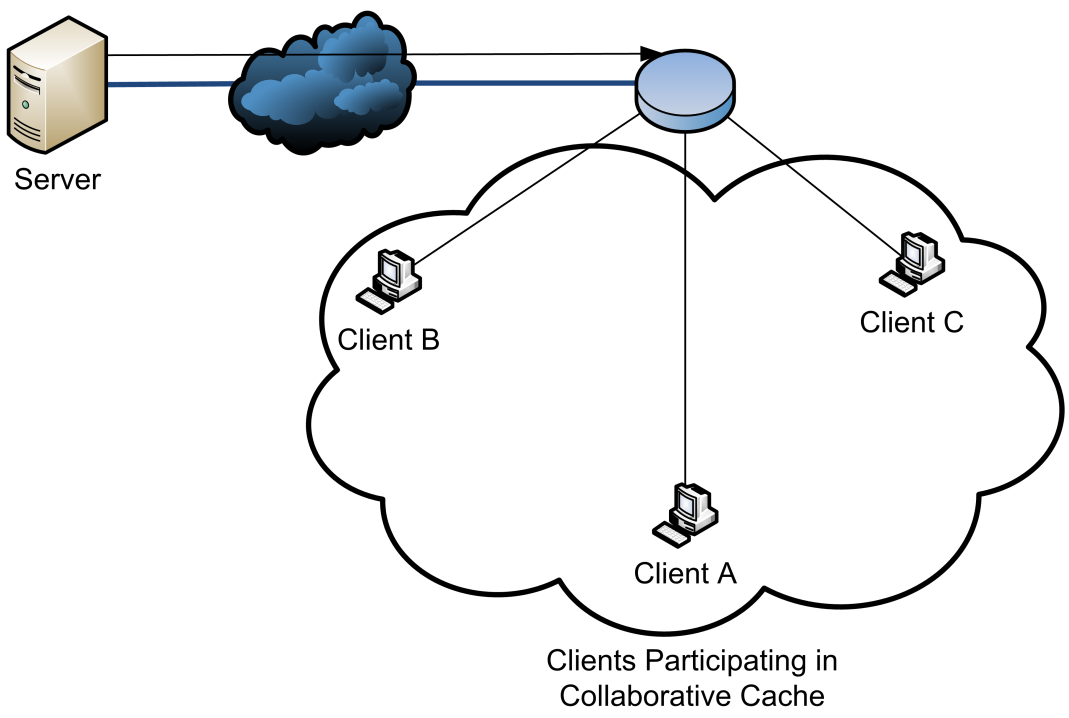


Figure 5.3: Peer Assisted Multicast Streaming

5.2 Behaviour

When using Peer Assisted Multicast Streaming a client normally receives content through two streams. One stream, serving the start of the content, the "prefix", which is served by another client in the collaborative cache, and the remainder of the content is supplied via a multicast stream from the server. As the server will not have to serve the remainder of the content to the client immediately, it has an opportunity to batch together multiple requests for the same content and serve them in the same multicast stream. If the prefix is not available in the collaborative cache then the server will stream the entirety of the content to the client. If content is never found in the collaborative cache the system will effectively behave as a familiar unicast system.

Figure 5.3 illustrates the behaviour of the architecture. The server delivers multicast streams, and the three clients participate in a collaborative cache to provide each other with unicast prefix streams.

In this example, Clients A, B and C all request the same stream at different times.

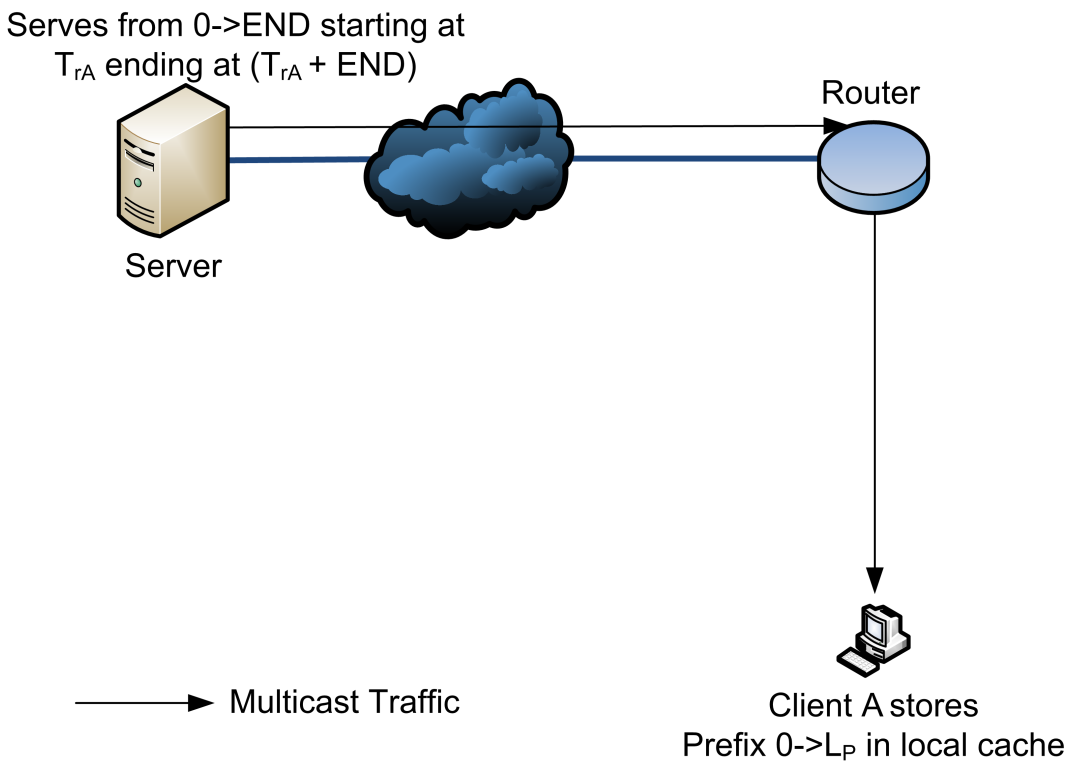

Serves from 0->END starting at T_{rA} ending at (T_{rA} + END)
Router
Server
Multicast Traffic
Client A stores Prefix 0->L_P in local cache

Figure 5.4: Delivery of content to Client A

Each client's cache is initially empty, which therefore means the collaborative cache is empty. Client A is the first to request the content, so is unable to locate the prefix in either its local cache or the collaborative cache. Client A will therefore need to retrieve the entirety of the content from the server. To do so, Client A sends a request for the entirety of the content to the server at time T_{rA}. This request requires immediate playback of content, so the server will not be able to service any other requests with the same stream. As soon as the server receives the request it will begin streaming the content to the client through a multicast stream. The setup and delivery of this stream is illustrated in Figure 5.4.

Once Client A has received the prefix of duration Lp, it will now be able to supply that prefix to other clients participating in the collaborative cache. Client B requests the content at time $T_{rB} > (T_{rA} + Lp)$. As the prefix is now stored in the collaborative cache by Client A, Client B requests the prefix from Client A. Once Client A has confirmed that it will be able to supply the prefix to Client B, Client B requests the

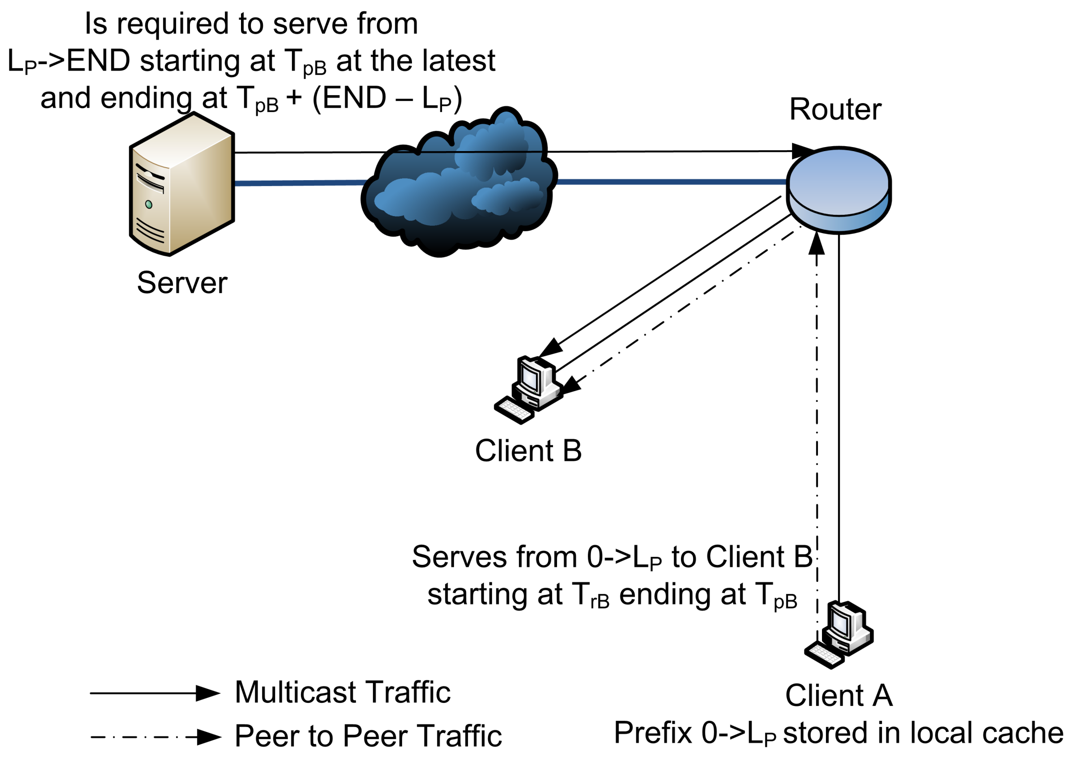

Is required to serve from
LP->END starting at TpB at the latest
and ending at TpB + (END – LP)
Router
Server
Client B
Serves from 0->LP to Client B
starting at TrB ending at TpB
Multicast Traffic
Peer to Peer Traffic
Client A
Prefix 0->LP stored in local cache

Figure 5.5: Delivery of content to Client B

remainder of the stream from the server. Client B informs the server that it requires the content from Lp until the end of the stream, and that the content must begin being received by T_{pB}. As the delivery of this stream by the server can be delayed by up to Lp, the server now has the opportunity to serve other requests for the same content with the same multicast stream. The setup and delivery of the content to Client B is illustrated in Figure 5.5.

At time $T_{rC} < T_{pB}$, Client C also requests the content. As the content is available in the collaborative cache the request sent to the server is identical to the request sent by Client B, except that in this case the stream needs to be served by T_{pC}. As there is already a stream pending for this content, the server can satisfy both T_{pB} and T_{pC} by serving a multicast stream at $MIN(T_{pB}, T_{pC})$. Client C will receive the stream from the server before it is required so the stream will need to be buffered and playback from the buffer at time can begin at T_{pC}. The setup and delivery of the content to Client C is illustrated in Figure 5.6.

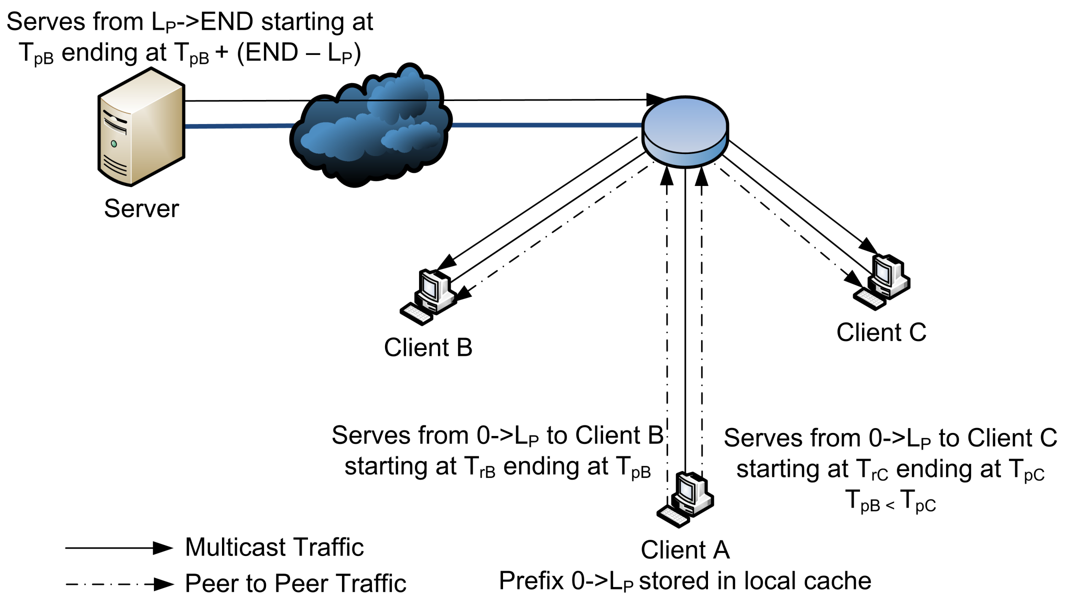

Serves from L_P->END starting at T_{pB} ending at T_{pB} + (END – L_P)
Server
Client B
Client C
Serves from 0->L_P to Client B starting at T_{rB} ending at T_{pB}
Serves from 0->L_P to Client C starting at T_{rC} ending at T_{pC} $T_{pB} < T_{pC}$
Client A
Prefix 0->L_P stored in local cache
Multicast Traffic
Peer to Peer Traffic

Figure 5.6: Delivery of content to Client C

The type of stream used to deliver the content to each client and the start and end times of these streams is illustrated in Figure 5.7.

In the following sections the behaviour of the collaborative prefix cache and of the multicast server will be described.

5.3 Collaborative Prefix Cache

As described in Section 4.1.2, the Squirrel [IRD02] collaborative web cache was described as a way of implementing a low-overhead, scalable web cache. Content is located in Squirrel using an overlay network, in this case Pastry [RD01]. The lookup request for a content item is routed through the peer-to-peer network to the *Home Node* whose identifier is numerically closest to the content item identifier. The authors of Squirrel explored two alternative approaches for storing cached content, which they termed *home-store* and *directory*.

In the *home-store* approach, each node is responsible for caching content items for which it is the *Home Node*. Thus, the cache at each node will contain both content items that were required locally and content items that were requested by other nodes

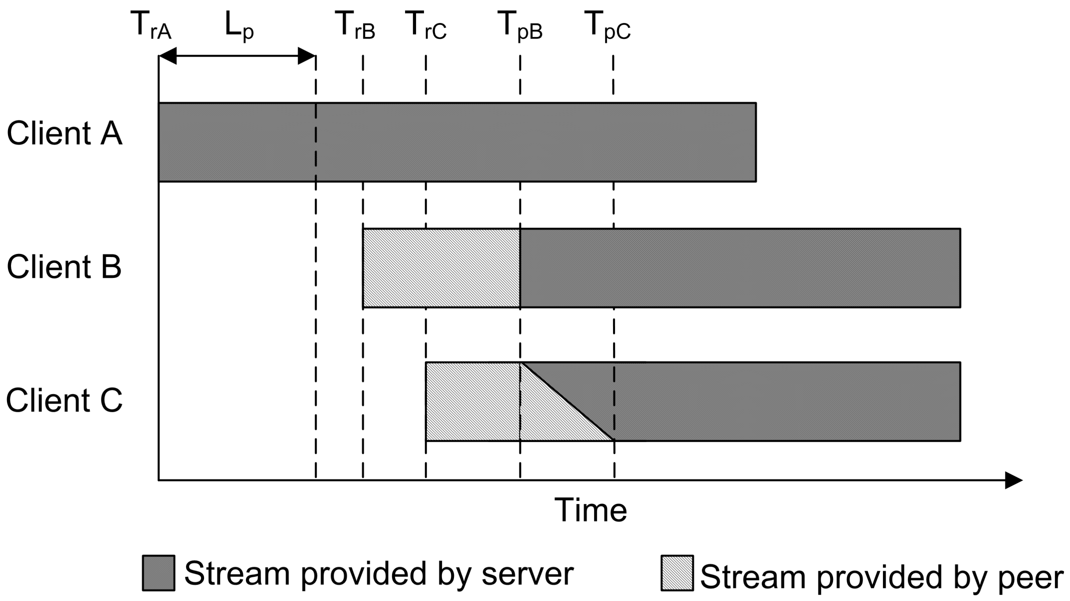


Figure 5.7: Clients B and C retrieve cached prefixes from Client A and the remainder of the stream from the server

and for which the node is the *Home node.*

If the *Home Node* does not have a copy of a requested item, it would be required to inform the requesting client that the item is not available and that the prefix must be requested from the main server. To prevent a subsequent failure to provide the content, the *Home Node* would need to obtain a copy of the item from the server.

Using the *directory* approach, *Home Nodes* are responsible for maintaining a directory of the nodes that have a cached copy of the content items whose identifiers are closest to the *Home Node's* identifier. The nodes in the directory are are termed *Delegates.* In the *directory* approach, each node only caches content items that were requested locally. Upon acquiring new content, each *Delegate* must notify the *Home Node* that it has a copy of the content.

In their original work on Squirrel, the authors established that *home-store* performed better than *directory* with respect to its ability to both reduce external bandwidth usage (resulting from cache misses) and also balance load internally among participating nodes. However, since caching and serving high bit-rate multimedia streams requires a significantly higher proportion of a client's bandwidth and storage resources than web content, it is unreasonable to expect a client to retrieve, store and serve content that it does not itself require. Accordingly, a *directory* approach similar to that proposed for Squirrel is more appropriate for a collaborative prefix cache for multimedia streaming applications.

Like the Squirrel decentralised web cache, clients in the proposed collaborative prefix cache collaborate in a peer-to-peer network to implement a cache of recently accessed content. Also like Squirrel, an overlay network has been used as the substrate for the collaborative cache. This approach exploits temporal locality in the same way as a centralised cache but avoids the need for additional infrastructure. The differences between the collaborative prefix cache and the original Squirrel approach arise from the need to store and supply potentially lengthy, high-bandwidth media streams.

Each client's role in the management of the collaborative cache will vary for different content items. In their role as *Delegates* clients are responsible for storing and supplying streams of prefixes and for managing the storage used by their local cache. In their role as *Home Nodes* clients are responsible for providing other clients with access to the *Delegates* of the content items for which it is responsible.

To ensure fairness each client is restricted to providing one concurrent prefix stream. This will ensure that load is evenly distributed within the collaborative cache. As each client may be a *Delegate* for a number of prefixes, each with a different *Home Node*, a *Home Node* will be unaware of whether a selected *Delegate* is already providing a prefix stream for a different content item. As a result *Delegates* must be able to reject requests to provide prefix streams. While this provides fairness, it also allows clients to renege on their responsibilities as *Delegates*. Preventing this disruptive behaviour is left for future work.

5.3.1 Prefix Retrieval

To provide a batching window to the server, a client must first locate a prefix of the content it is requesting. If there is no prefix in the client's local cache, then the client will try to locate a prefix in the collaborative cache.

To locate a copy of the prefix the client must send a request to the *Home Node* responsible for the content it is requesting. As a *Directory* approach is used, the *Home Node* is responsible for maintaining a directory of the *Delegates* that have a cached copy of the prefix. Upon receiving the request, the *Home Node* must select a *Delegate* and forward the request to the *Delegate*. If the *Delegate* has a valid[1] copy of the content and is not currently providing another prefix stream, the *Delegate* will stream the content to the requesting client. If the *Delegate* is unable to provide the prefix, it informs the *Home Node*, which must try to select another *Delegate*. If no suitable *Delegates* exist, then the *Home Node* must inform the requesting client that the content item is not available in the collaborative cache and that it must obtain a copy of the item from the server.

5.3.2 Prefix Announcement

Upon obtaining a copy of a prefix, because a *Directory* approach is used, a client must notify the *Home Node* for the content item that the client is now a potential *Delegate*. In the original Squirrel approach the timing of this notification was not essential as the web content being cached only took a short time to be fully downloaded by the client. Due to the amount of time required to receive and cache multimedia content, however, the timing of this notification is significant. Two approaches for announcing new content are proposed, in the first, the *Delegate* will only inform the *Home Node* once the complete prefix has been cached. In the second approach, the *Delegate* will notify the *Home Node* as soon as caching of the content begins, and if required the new *Delegate* can supply an incomplete prefix to a requesting client. This should result in more content being retrieved from the collaborative cache in the event of new popular content being introduced to the system. This effect is discussed in Section 7.9.

The main advantage of using only complete prefixes is that requesting clients will always receive the complete prefix from the collaborative cache, which should allow the batching window provided to the server to be maximised. There is also the advantage

[1]The validity of cached prefixes will be discussed in Section 5.3.3

of minimising traffic between the *Delegate* and the *Home Node*, as only notifications of content addition and removal will be required between the two nodes. The main disadvantage of using complete prefixes though, is that the service responds slowly to the addition of new popular content as the entire prefix must be cached by each *Delegate* before it can be provided to another client. As few prefixes will be available initially, the server will need to serve the entirety of a content item for many of the requests for the new content without the opportunity of batching together these requests. This issue is illustrated and discussed in detail in Chapter 7 where the two approaches will be compared.

For the system to use incomplete prefixes, the *Delegate* will be required to notify the *Home Node* as soon as the start of the prefix has been received. A further notification is sent to the *Home Node* upon caching the entirety of the prefix. If a *Delegate* is required to provide an incomplete prefix to another client, it will only be required to provide the portion of the prefix it has already received. Although the *Delegate* may receive more of the prefix while it is delivering the incomplete prefix, this is not guaranteed. By agreeing to provide only the portion of the prefix that was cached when the request was received, cascading failures can be avoided.

The *Home Node* must also be notified upon removal of any prefix by a *Delegate*.

5.3.3 Prefix Validation

Cached prefixes can become out of date with respect to the current version of the content. This may be due to alterations to the original content. As with Squirrel, the PAMS collaborative cache allows for the validation of prefixes content without always having to check the validity of the prefix against the origin server. This will avoid the need to perform additional communication with the central server.

To avoid excessive validation of prefixes, a Time To Live (TTL) is associated with the metadata of each cached prefix. If the gap between the current time and time the prefix was last validated is greater than the TTL, then the content is deemed to be stale, and will need to be validated, otherwise the prefix is considered fresh.

To determine whether content has been altered, a last-modified timestamp is associated with each content item. This timestamp allows for a *Delegate* to determine if its copy of the prefix is up-to-date with respect to the original content item. If the last-modified timestamp is older than the timestamp at the central server, then the cached prefix is no longer valid and must replaced.

Both clients and *Delegates* will occasionally need to validate the prefixes in their caches but only when a prefix is stale, so as to avoid unnecessary communication. As the *Home Node* is the central point of management for the prefix, a client should validate a prefix in its local cache against the *Home Node* in the collaborative cache. The *Delegates* though, will need to validate prefixes against the origin server. To prevent multiple validation messages being directed at the server, the *Delegate* that performs the validation, will forward on the response to the *Home Node*, which will forward on the information to other *Delegates* as required.

As prefix validation only needs to be done when the prefix is required for playback, validation is integrated into the prefix retrieval approach described in Section 5.3.1. As a result a client will always contact the *Home Node* first, whether it is checking the validity of its own cached prefix, or attempting to retrieve a prefix from the collaborative cache.

The following scenarios apply to the retrieval and validation process. In each case, a prefix may be stored in the local cache of a requesting client or stored in the collaborative cache. Furthermore, a stored prefix may be be fresh if its TTL has not expired or it may be stale, in which case it must be validated against the collaborative cache or central server. The validation process will determine whether the original content item has been modified or not modified.

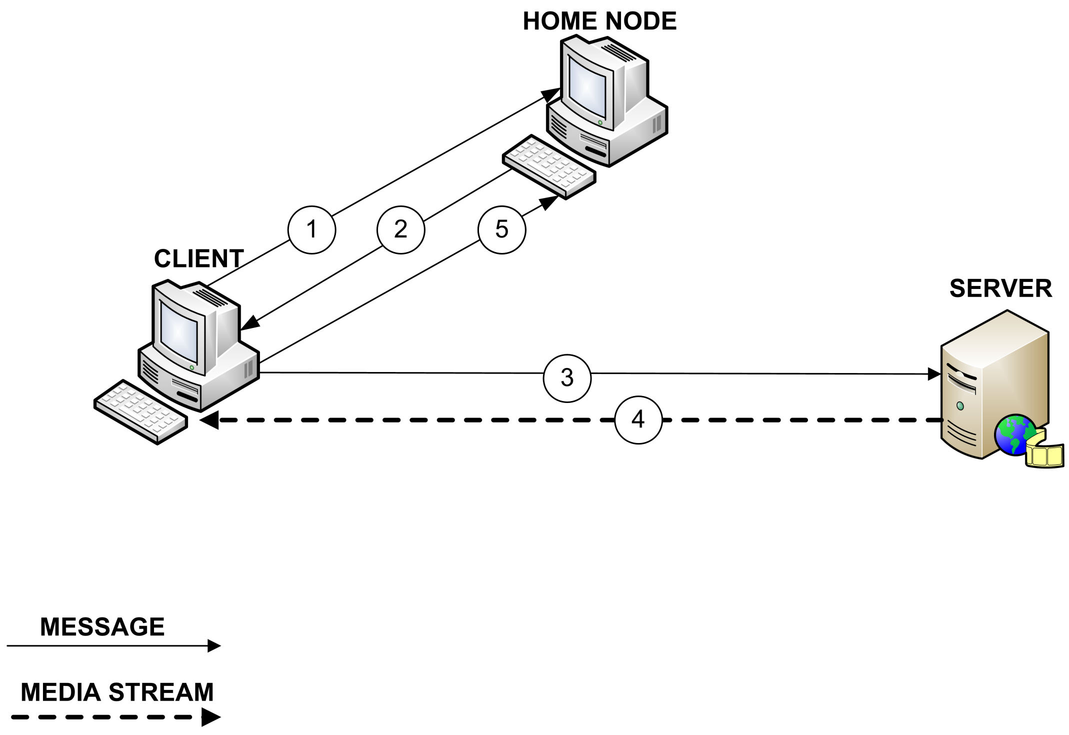


Figure 5.8: Case #1: Not in local cache, not in collaborative cache

Case #1: Not in local cache, not in collaborative cache

This scenario is illustrated in Figure 5.8.

1. The client does not have a copy of the prefix in its local cache, so sends a retrieval message to the *Home Node*
2. The *Home Node* is unaware of the content, or has no available *Delegates*, and sends a message back to the client to indicate this
3. The client sends a retrieval request to the server
4. The server provides the entire content stream to the client
5. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

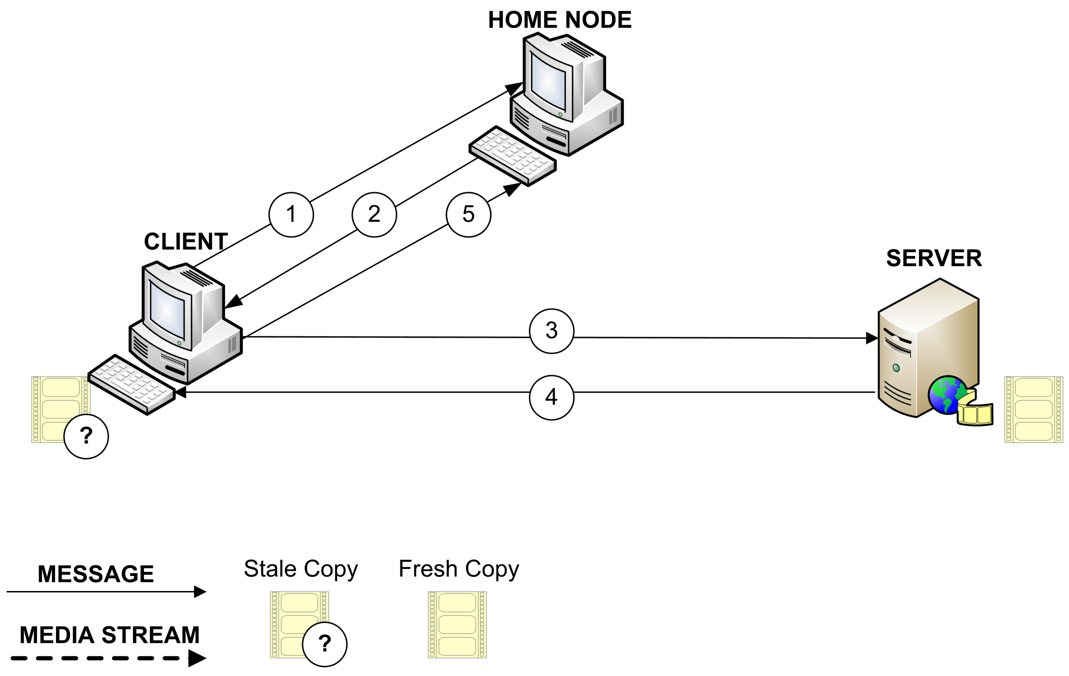


Figure 5.9: Case #2: Stale in local cache, not in collaborative cache, not modified

Case #2: Stale in local cache, not in collaborative cache, not modified

This scenario is illustrated in Figure 5.9. This case can only arise if the *Home Node* is unaware of existing *Delegates* of the content.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* is unaware of the content, or has no available *Delegates*, and sends a message back to the client to indicate this
3. The client sends a validation request to the server
4. The server validates the content as being not modified and the client uses its local copy of the prefix
5. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

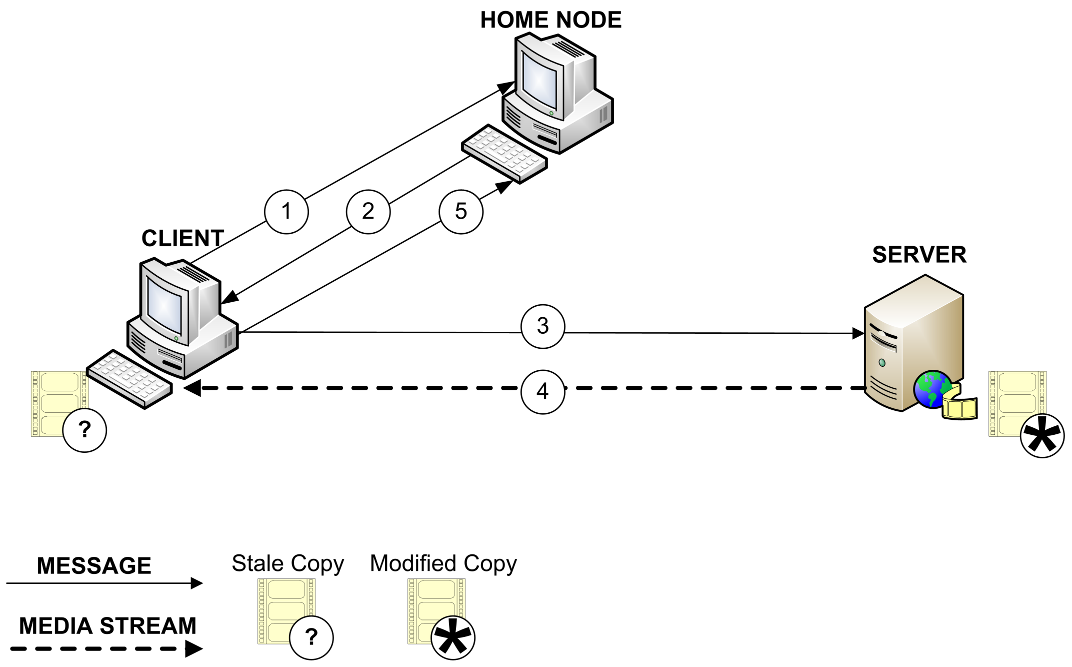


Figure 5.10: Case #3: Stale in local cache, not in collaborative cache, modified

Case #3: Stale in local cache, not in collaborative cache, modified

This scenario is illustrated in Figure 5.10. This case can only arise if the *Home Node* is unaware of existing *Delegates* of the content.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* is unaware of the content, or has no available *Delegates*, and sends a message back to the client to indicate this
3. The client sends a validation request to the server
4. The server verifies the content as modified and the server provides the entire content stream to the client
5. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

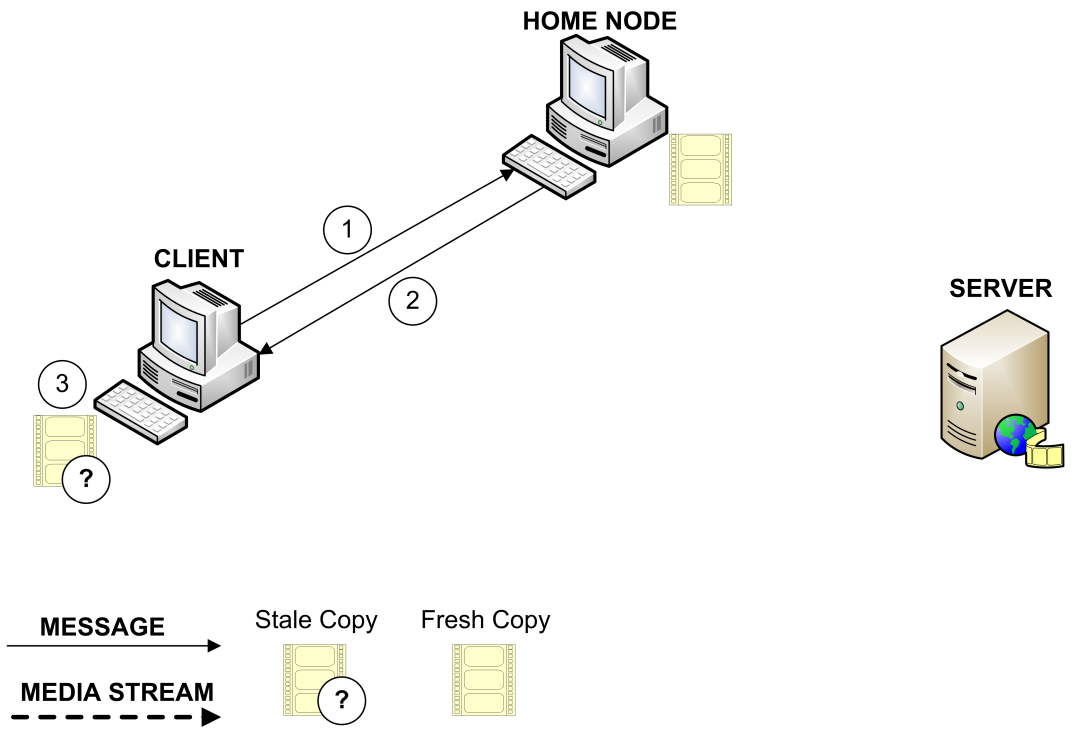


Figure 5.11: Case #4: Stale in local cache, fresh in collaborative cache, not modified

Case #4: Stale in local cache, fresh in collaborative cache, not modified

This scenario is illustrated in Figure 5.11.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* is aware of the content, and validates the content as not having been modified
3. The client uses its locally cached copy of the prefix

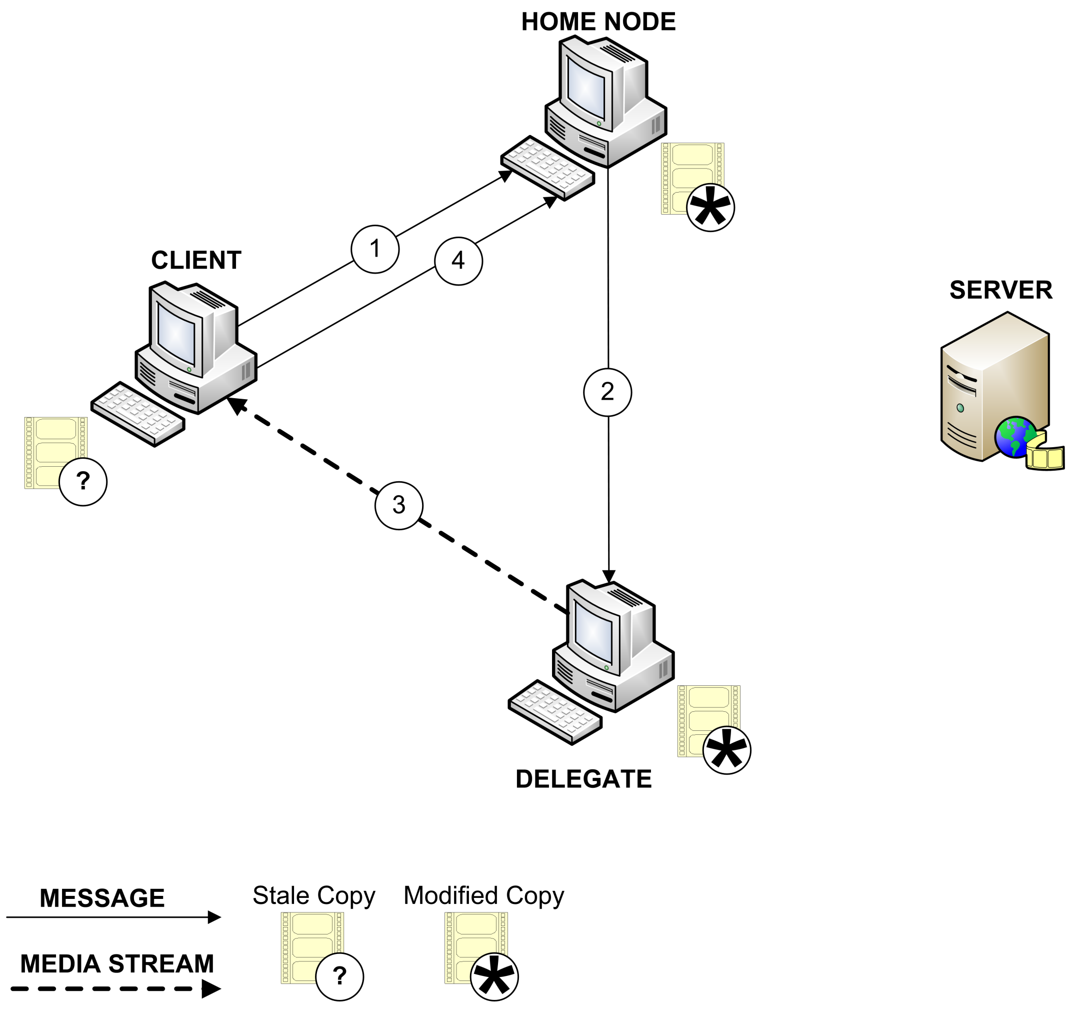


Figure 5.12: Case #5: Stale in local cache, fresh in collaborative cache, modified

Case #5: Stale in local cache, fresh in collaborative cache, modified

This scenario is illustrated in Figure 5.12.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* is aware of the content, and identifies the content as modified. The *Home Node* forwards the request to a selected *Delegate*
3. The *Delegate* streams the modified prefix to the client

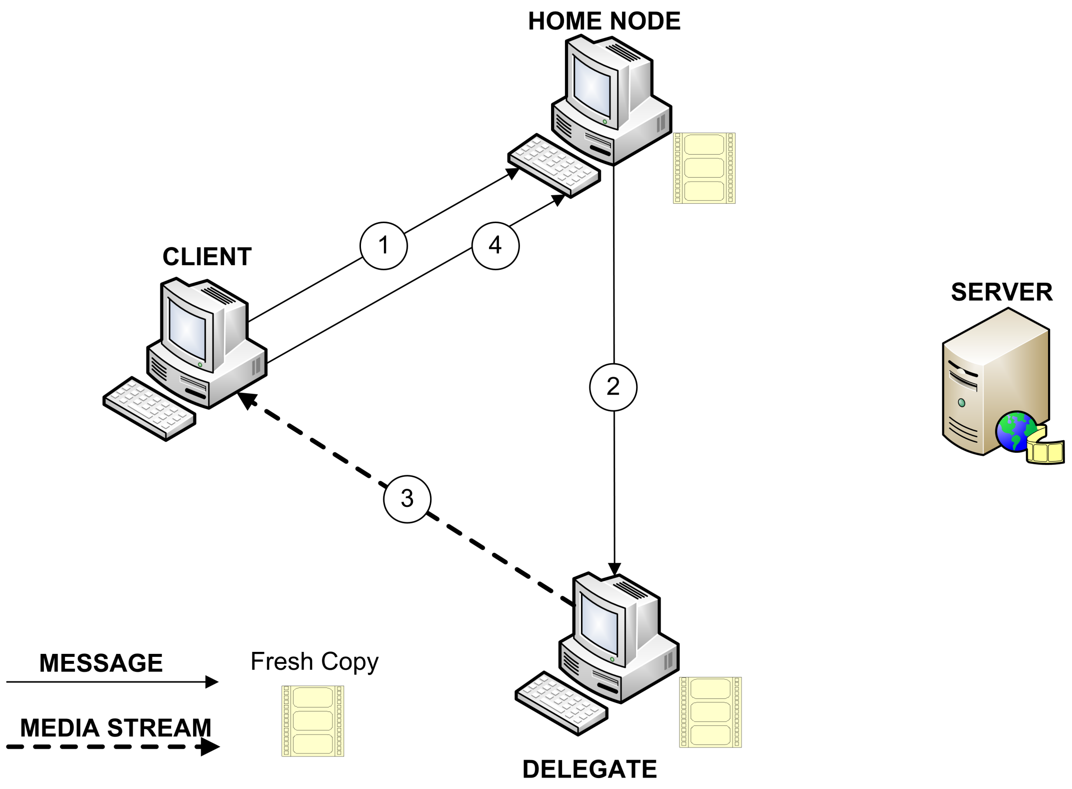


Figure 5.13: Case #6: Not in local cache, fresh in collaborative cache

4. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

Case #6: Not in local cache, fresh in collaborative cache

This scenario is illustrated in Figure 5.13.

1. The client does not have a copy of the prefix in its local cache, so sends a retrieval message to the *Home Node*
2. The *Home Node* forwards the request to a selected *Delegate*
3. The *Delegate* streams the prefix to the client
4. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

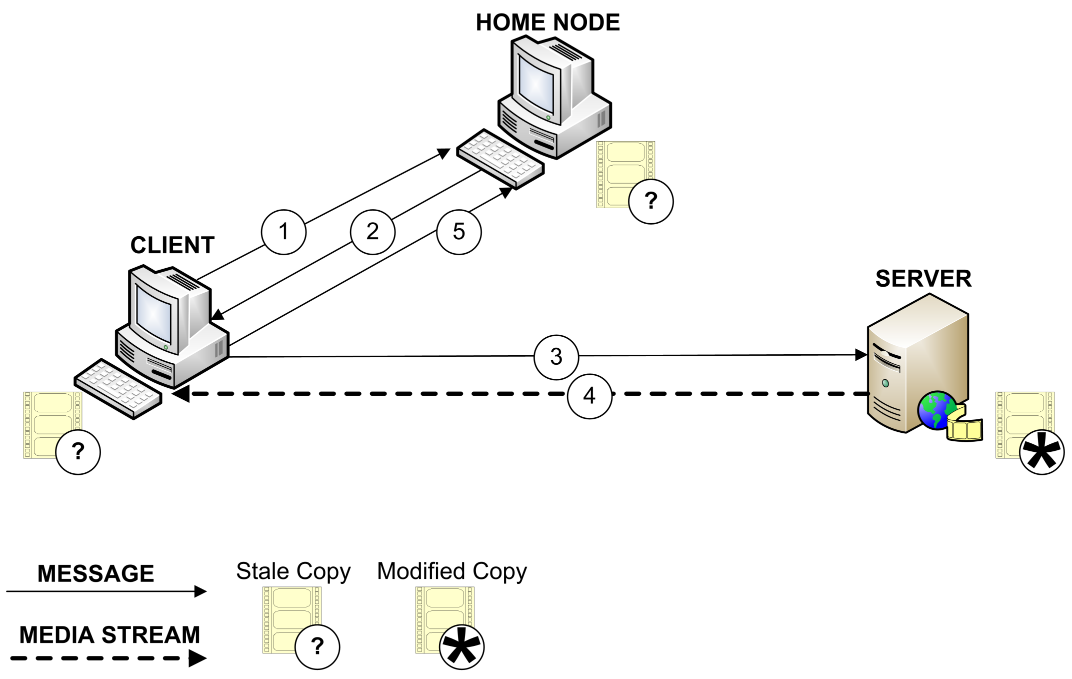


Figure 5.14: Case #7: Stale in local cache, stale in collaborative cache, modified

Case #7: Stale in local cache, stale in collaborative cache, modified

This scenario is illustrated in Figure 5.14.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* sees the content as stale, so cannot validate the content in its response to the client
3. The client sends a validation message to the server
4. The server indicates that the content has been modified and streams the prefix to the client
5. The client will notify the *Home Node* that it is a potential *Delegate* and will update the *Home Node's* last-modified timestamp

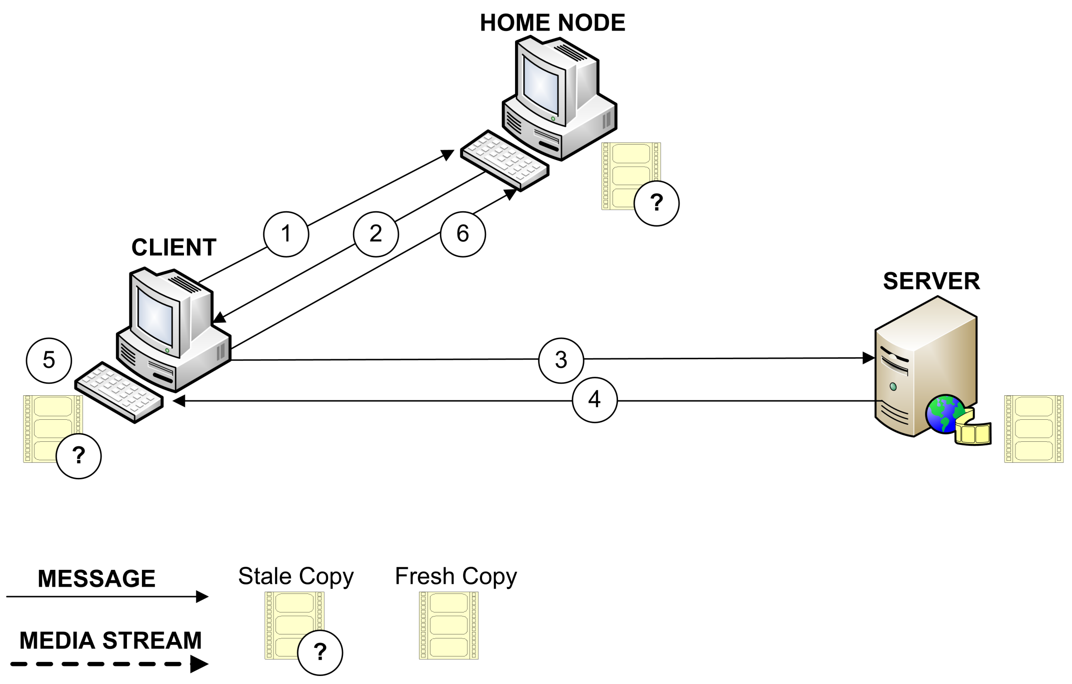


Figure 5.15: Case #8: Stale in local cache, stale in collaborative cache, not modified

Case #8: Stale in local cache, stale in collaborative cache, not modified

This scenario is illustrated in Figure 5.15.

1. The client has a copy of the prefix in its local cache which is stale, so sends a validation message to the *Home Node*
2. The *Home Node* sees the content as stale, so cannot validate the content in its response to the client
3. The client sends a validation message to the server
4. The server validates the prefix as having not been modified
5. The client uses its locally cached copy of the prefix
6. The client will notify the *Home Node* that it is a potential *Delegate* and will refresh the *Home Node's* TTL for the directory entry

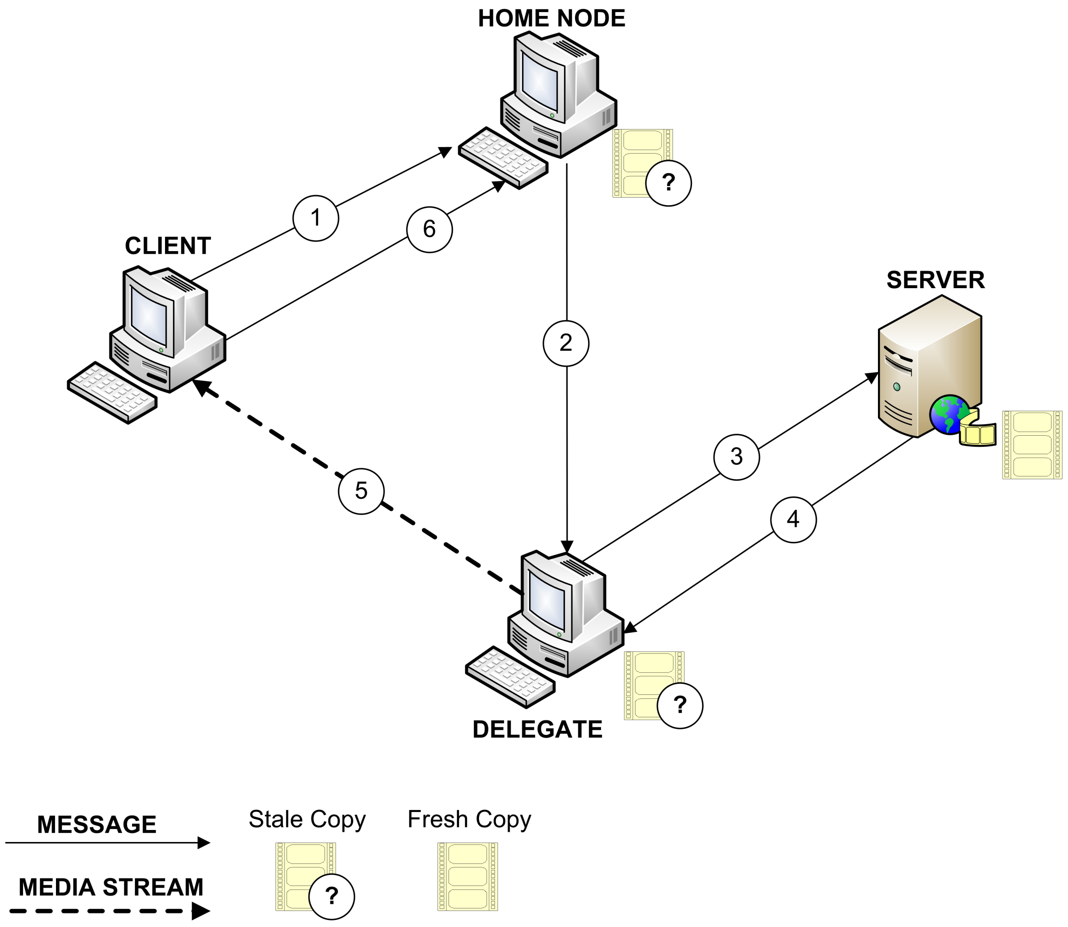


Figure 5.16: Case #9: Not in local cache, stale in collaborative cache, not modified

Case #9: Not in local cache, stale in collaborative cache, not modified

This scenario is illustrated in Figure 5.16.

1. The client does not have a copy of the prefix in its local cache, so sends a retrieval message to the *Home Node*
2. The *Home Node* views the content as stale and forwards the request to a selected *Delegate*
3. The *Delegate* sends a validation message to the server
4. The server responds that the content has not been modified
5. The *Delegate* streams the prefix to the client and concurrently refreshes the *Home*

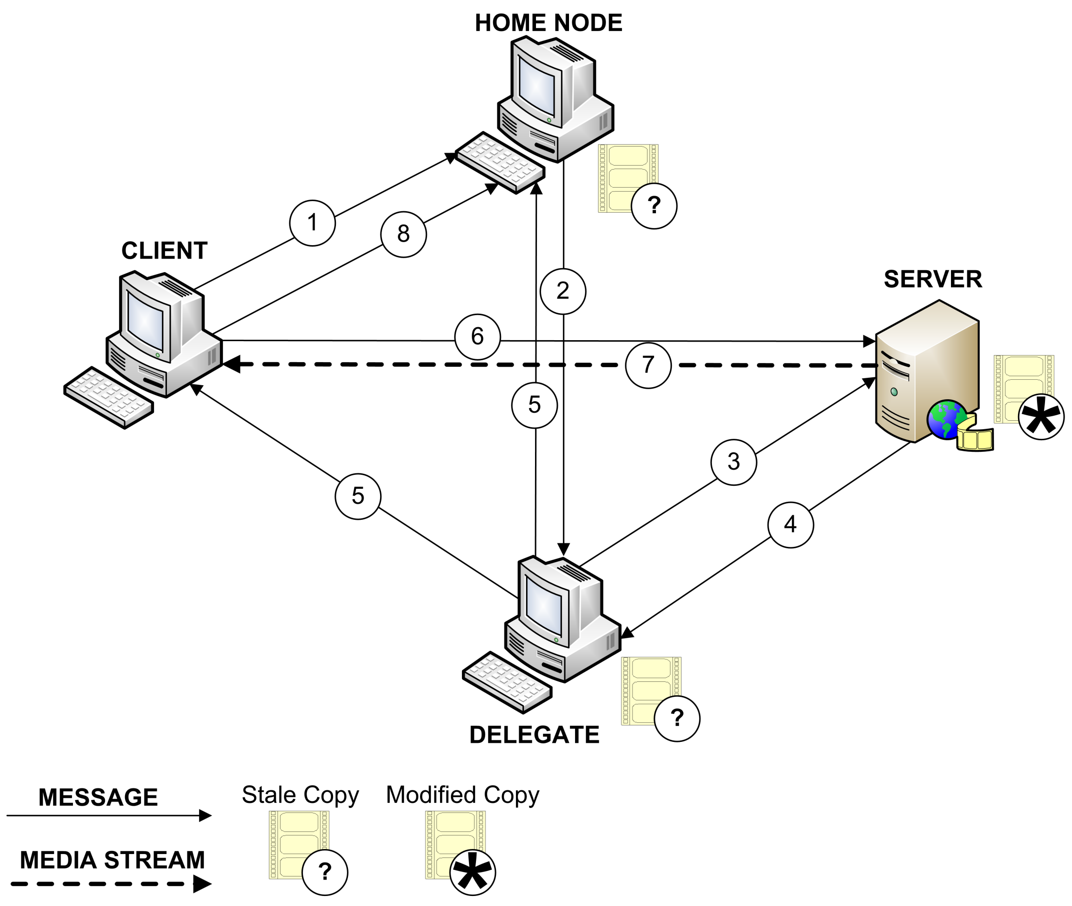


Figure 5.17: Case #10: Not in local cache, stale in collaborative cache, modified

Node's TTL for the directory entry

6. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

Case #10: Not in local cache, stale in collaborative cache, modified

This scenario is illustrated in Figure 5.17.

1. The client does not have a copy of the prefix in its local cache, so sends a retrieval message to the *Home Node*

2. The *Home Node* views the content as stale and forwards the request to a selected *Delegate*

3. The *Delegate* sends a validation message to the server

4. The server responds that the content has been modified

5. The *Delegate* sends a message to the client responding that the content is not available, removes it's out of date version of the content and updates the *Home Node* with the new last-modified time

6. The client sends a retrieval request to the server

7. The server streams the prefix to the client

8. The client will notify the *Home Node* that it is a potential *Delegate* for the prefix

The trivial case where a client has a fresh cached copy of a prefix has been omitted. The complete lookup decision procedure for clients, which shows how clients interact with the collaborative cache (C/C), is detailed in Figure 5.18. The actions taken by the *Home Node* and by a selected *Delegate* upon receiving a request are detailed in Figure 5.19.

5.3.4 Delegate Selection

The *Home Node* for each content item is responsible for selecting a *Delegate* in response to client requests. Upon receiving a request for content, this request must be forwarded onto a *Delegate*. The *Delegate* should be chosen in a way that maximises the efficiency of the system, but is also fair to the clients contributing to the collaborative cache.

As discussed in Section 5.3.2 complete prefixes or a combination of complete and incomplete prefixes can be provided by the *Delegates*. For maximum benefit to the server, incomplete prefixes are only used if there are no complete prefixes available, as this should provide the largest possible batching window to the server.

For complete prefix selection, a round robin *Delegate* selection policy will ensure fairness.

For incomplete prefixes, a round robin selection policy would ensure fairness but could result in the use of very short prefix streams, which would offer very little

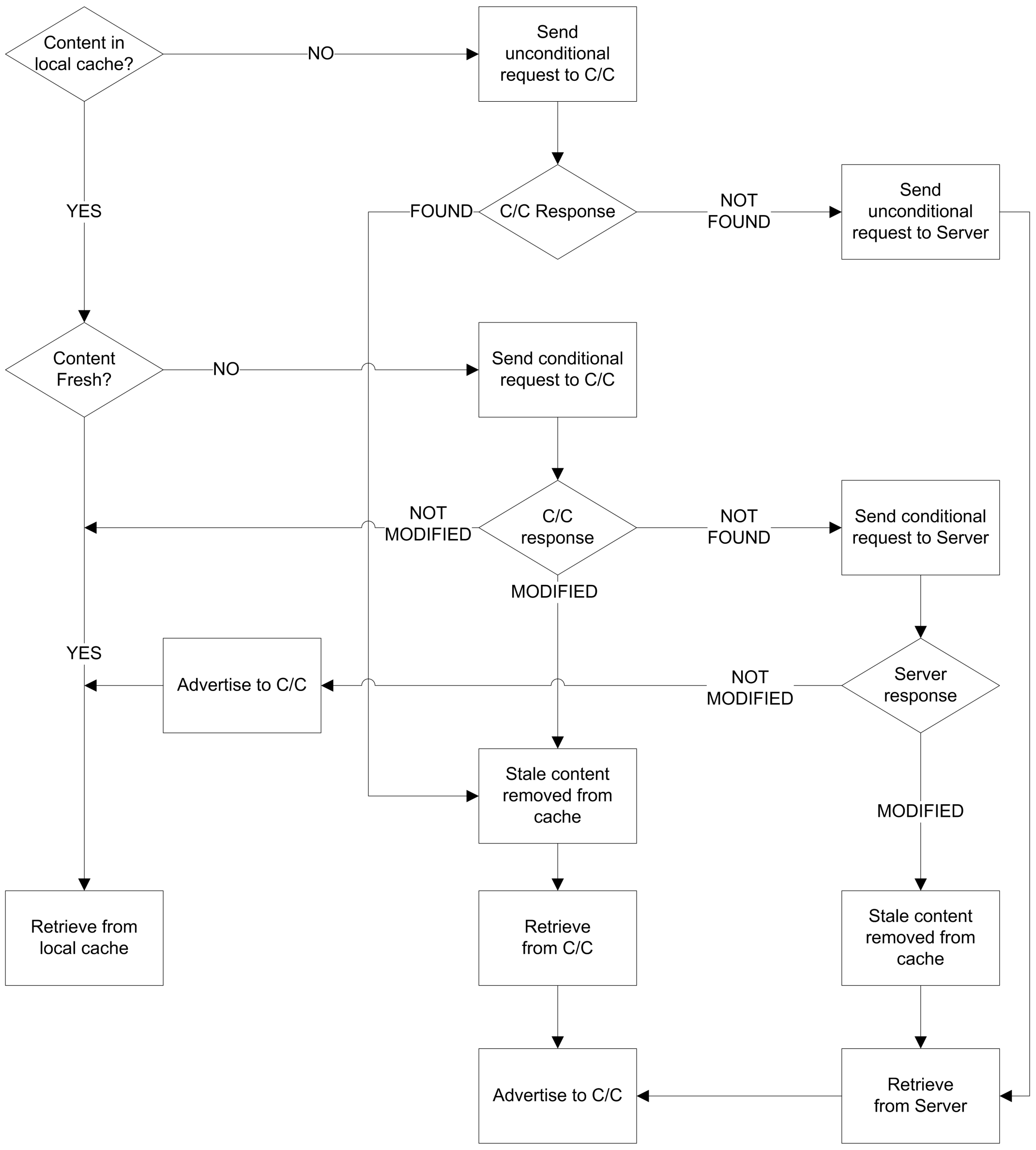

Content in local cache?
NO
Send unconditional request to C/C
YES
FOUND
C/C Response
NOT FOUND
Send unconditional request to Server
Content Fresh?
NO
Send conditional request to C/C
NOT MODIFIED
C/C response
NOT FOUND
Send conditional request to Server
MODIFIED
YES
Advertise to C/C
NOT MODIFIED
Server response
Stale content removed from cache
MODIFIED
Retrieve from local cache
Retrieve from C/C
Stale content removed from cache
Advertise to C/C
Retrieve from Server

Figure 5.18: Client Decision Flow Chart

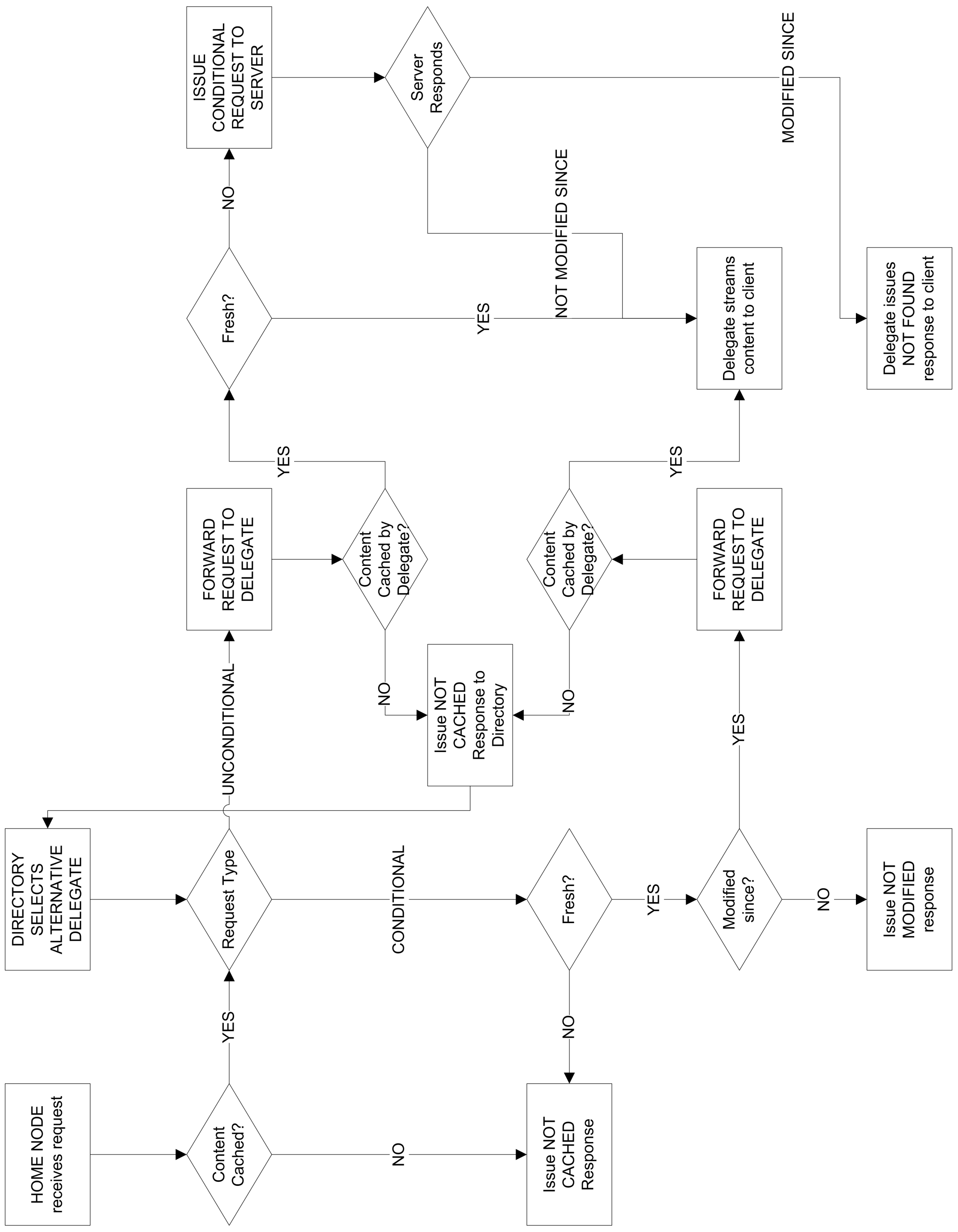

HOME NODE receives request
Content Cached?
NO
Issue NOT CACHED Response
YES
Request Type
DIRECTORY SELECTS ALTERNATIVE DELEGATE
CONDITIONAL
Fresh?
NO
YES
Modified since?
NO
Issue NOT MODIFIED response
YES
FORWARD REQUEST TO DELEGATE
Content Cached by Delegate?
NO
Issue NOT CACHED Response to Directory
YES
Delegate streams content to client
UNCONDITIONAL
FORWARD REQUEST TO DELEGATE
Content Cached by Delegate?
NO
YES
Fresh?
NO
ISSUE CONDITIONAL REQUEST TO SERVER
Server Responds
NOT MODIFIED SINCE
MODIFIED SINCE
YES
Delegate issues NOT FOUND response to client

Figure 5.19: Home Node and Delegate Decision Flow Chart

batching opportunity to the server. The longer the prefix a *Delegate* can provide, the more beneficial it is to the system as a whole so the longest incomplete prefix should be selected by the *Home Node*. This can be established based on the time the *Delegate* notified the *Home Node* that the *Delegate's* cache contained an incomplete prefix. To reduce the complexity of the *Home Node* the *Delegates* that notify the *Home Node* first are assumed to have the longest, or most complete prefixes. In order to make a more correct decision, the *Delegate* would need to regularly update the *Home Node* on the state of its incomplete prefix, or the *Home Node* would need to query a number of *Delegates* when choosing a *Delegate* to serve the incomplete prefix. This would increase the number of control messages generated by PAMS and would increase the bandwidth utilisation of the *Home Nodes* and *Delegates*. There would also potentially be an increase in the time taken for the *Home Node* to select a *Delegate*.

5.3.5 Local Cache Management

Each client's cache capacity is finite, and as such, when a client is requesting new content, a prefix of another content item will need to be replaced if the local cache is full. The decision of which item to be removed requires consideration as the removal of content will have an effect on the performance of collaborative cache. Ideally the client will make the best possible decision for the performance of the collaborative cache and the system as a whole. However, this is difficult due to the distributed nature of the information relating to the content items.

If it is assumed that the most popular content in the past will be of most benefit to the system in the future a client should only remove the least popular content from its cache. The distributed nature of the collaborative cache, however, makes it difficult to establish the relative popularity of different content items.

For a content item, only the corresponding *Home Node* and the server will be aware of how frequently the content item is requested, as each *Delegate* will only receive a small portion of all the requests for the content. As a result a *Delegate* cannot independently relate the popularity of the content items stored in its local cache and cannot therefore establish the best candidate for removal. While the client could request this information from the server for all the items in its cache and make a decision based on this information, this would be going against one of the principles the system is based on, which is to reduce network traffic around the server.

A more decentralised approach would be to individually poll each of the *Home Nodes* for each content item in the local cache, requesting the system wide frequency of access. This approach though, would be very costly for large cache sizes, as for a cache size of N prefixes, $2N$ messages would be required to request the information and receive the response. Furthermore, the request messages would need to be routed through the overlay network. Also, the decision of which prefix to remove would have to be delayed until all responses have been received. One of the guiding principles of PAMS is that it should be lightweight, and this approach would violate this principle as large amounts of communication between nodes would be performed frequently.

Therefore a client should make decisions based solely on the information the client has at the time the decision is required. This decision can be based on information generated by accesses to the local cache. There are two ways in which the cache can be accessed, either locally by the client itself or remotely, when the cache is accessed for the purpose of serving a prefix to another client. The access frequency can be used to relate the popularity of each cached prefix and this can be used as an approximation to establish the importance of the content to the system. The item which has the lowest access frequency is the item that should be replaced.

Local and remote access frequencies can be combined in three approaches to evaluate the popularity or importance of each prefix. In a selfish approach the client will only use its own accesses to establish the popularity of the content. An altruistic approach would consider only remote accesses to the clients cache when evaluating popularity. The third approach combines local and remote accesses.

The selfish approach prevents the collaborative cache taking priority over local needs. This can be important, as it should prevent the client from being forced to remove content that is locally accessed frequently, and therefore repeatedly having to re-acquire the prefix from the collaborative cache.

While a client having to re-acquire content should be avoided, the cost of this to the overall system is negligible compared to the the cost of not prioritising the provision of the most popular content. As such, the altruistic approach should result in the collaborative cache attaching greater importance to accesses from the collaborative cache. As there are advantages to both approaches, a combined approach should also be considered. In practice, however, performance evaluations have shown there to be a negligible difference in the performance resulting from the three approaches.

5.3.6 Joining the Collaborative Cache

Upon joining the collaborative cache, a node must take up its potential roles as a *Delegate* for the content stored in its local cache and as a *Home Node*. The node must announce any pre-existing prefixes stored in its own local cache to each of the relevant *Home Nodes* in order to be a considered as a potential *Delegate* in the collaborative cache.

The joining node must also take on the role of *Home Node* for the content items the node may now be responsible for. In order to discover what content items the joining node is now responsible for, the joining node will need to contact nodes nearby, as these nodes will determine and send on the directory entries for these content items. The content for which the joining node will be responsible for will be determined by the overlay network and the implementation of this is discussed in Section 6.1.1.

5.4 Server

The server may be required to supply the entire content stream or the remainder of the content if the client is successful in obtaining a prefix. In order to allow for the server to properly assign and schedule multicast streams, the client must specify which content item it is requesting, which point within the content the stream should begin at and at what time the stream is required by the client.

If the client was unsuccessful in retrieving a prefix for a content item, it will request the entire content stream from the server, with an offset of zero to begin immediately, at the current time T. The server will immediately serve a multicast stream of the content.

If the client was successful in retrieving a prefix of duration Lp, it will request the remainder of the content, with an offset of Lp, to be served no later than $T + Lp$. Upon receipt of this request, if the server has an already scheduled, or is already serving, a multicast stream for that content item, that will serve from offset Lp before time $T+Lp$, then the request may be assigned to this stream. Otherwise, if a stream is scheduled to serve from offset Lp but at a time after $T + Lp$, then the request may be assigned to this stream and the start time will be brought forward to serve offset Lp at time $T+Lp$. It may be possible to modify the offset of a scheduled stream to begin at an earlier offset, once doing so will still satisfy the requirements of all previous requests assigned to that stream. Otherwise a new stream will be scheduled to serve from offset Lp at time $T + Lp$. While synchronisation will not be precise between the overlapping of the prefix the client is receiving and the multicast stream from the server, this approach should yield minimal impact on the client, and any incorrect estimation will not result in a total playback failure.

5.5 Summary

This chapter has described the design of the proposed Peer Assisted Multicast Streaming architecture. This architecture combines the use of a collaborative cache based on an overlay network with a multicast-capable multimedia streaming server. The aim of the architecture is to improve the performance of an on-demand streaming application.This is achieved by using the collaborative cache to provide prefixes of requested content items, thereby offering the server the opportunity to serve multiple requests for a content item with a single multicast stream. This is done while still providing clients with fully interactive control over streams.

This chapter has focused on the design on the collaborative prefix cache and has also described the policy to be implemented by the multicast server. Chapter 6 will propose how the PAMS architecture can be implemented using standard network protocols and an existing overlay network. The performance of PAMS will be evaluated in Chapter 7.

Chapter 6

Implementation of Peer Assisted Multicast Streaming

Chapter 5 described in detail the proposed Peer Assisted Multicast Streaming, or PAMS, architecture. In this chapter, an approach for implementing PAMS is described. Firstly, an implementation of a collaborative prefix cache for PAMS using an existing peer-to-peer overlay network is proposed, including an approach for implementing prefix streaming from *Delegates* to clients participating in the collaborative cache using a standard streaming control protocol. Secondly, an approach for implementing multicast streaming from servers to clients is described, again using a standard stream control protocol. A model of the proposed PAMS implementation, developed using a detailed network simulation framework, is presented at the end of the chapter.

6.1 Implementing the Collaborative Prefix Cache

Many existing overlay networks would be appropriate for implementing PAMS. The proposed implementation described in this thesis uses the Chord [SMK$^+$01] overlay network, which is similar to Pastry upon which the Squirrel collaborative web cache was constructed. The structure of Chord and how it can be used to route messages within an overlay network is described in Section 4.1.

6.1.1 Joining and Leaving the Collaborative Cache

Regardless of which overlay network technology is used, the roles and associated responsibilities in the architecture will be the same but the precise implementation of those responsibilities may differ. The actions that must be performed by a node upon joining the collaborative cache have been described in Section 5.3.6 and the implementation of those actions is described here.

On joining the collaborative cache, a node will have to perform actions relating to its role as a potential *Delegate* and as a potential *Home Node*. In order to be considered as a *Delegate*, the node will need to announce the contents of its own local cache. To do this it must create an overlay network message for each of its prefixes (be they complete or incomplete) and send the messages to the *Home Node* responsible for each entry.

When a node joins the collaborative cache, it may now be the *Home Node* for particular content items, as the identifier of the new node may now be closer to that of one or more content items than any pre-existing node. If a Chord based system is used to implement PAMS, the joining node will need to send a Chord message to the node with the next highest numerical identifier (this information is provided by the Chord overlay network). The node receiving this message can determine which content items the new node is responsible for and send the directory entries back to the new node so that it can assume the role of *Home Node* for the content.

(If a Pastry based overlay network was in use, then the joining node would need to contact the next highest and lowest nodes in the network. Again these nodes would determine which content items the joining node is now responsible for, and send the relevant information to the joining node.)

As nodes can leave the network suddenly, there is no mechanism for allowing a *Home Node* to send its directory listings just before leaving the network. To avoid losing this information, periodic replication of directory listings is required. Furthermore, as the philosophy proposed by this thesis is to adopt a lightweight approach to the implementation of the collaborative cache, nodes that leave the collaborative cache in a structured way do not announce their departure in advance as, if replication is used as described below, there will not be a significant loss of information.

6.1.2 Replication

In order to prevent directory information being lost when nodes depart from the collaborative cache, *Home Nodes* can replicate their directory contents to other *Home Nodes*. Without replication, the departure of a *Home Node* will remove from the collaborative cache all knowledge of existing prefixes of content items that mapped to the departing node. These prefixes, will still be stored by the respective *Delegates*, but no requests for the prefix will be directed to them.

To prevent the loss of directory information within the collaborative cache, a list of *Delegates* for that content, and the associated metadata of the content will need to be replicated in the overlay network. As the most popular prefixes will have a large number of *Delegates*, it is not vital that replication maintains an exact copy of the directory information, allowing replication to be infrequent.

The current *Home Node* for each content item can determine the node that will become the new *Home Node* for the content item should the current node leave the network. Each *Home Node* can therefore send periodic messages to replicate the *Delegate* information for each content item to the node that would become the new *Home Node* for the content.

6.1.3 Implementing Prefix Streaming with RTSP

Protocols are required to both control and transport multimedia streams. A well defined and accepted standard for this is the Real Time Streaming Protocol (RTSP) [SRL98], which has been described in Chapter 2. This protocol and its variants are used by commercial applications including QuickTime, Windows Media Player and RealPlayer, making it an obvious candidate for the implementation of Peer Assisted Multicast Streaming.

As described previously, clients receiving a multimedia content stream using PAMS typically receive two streams: a prefix stream containing the start of the content from another client, or *Delegate*, participating in the collaborative cache and a second multicast stream containing the remainder of the content from a server. This section describes a proposed approach for implementing prefix streaming from collaborative cache *Delegates* to requesting clients.

When providing a prefix, a *Delegate* behaves like a simple unicast multimedia server, serving the prefixes available in its local cache to requesting clients. The clients will have been directed to the *Delegate* by the *Home Node* responsible for the requested content. The RTSP stream control protocol is proposed for both controlling prefix streams provided by *Delegates* and for controlling the multicast streams provided by the server.

Before a client requests a prefix from the collaborative cache, it will check whether the prefix is available in its local cache. Regardless of whether a required prefix is located in the local cache or collaborative cache, the cached prefix will need to be validated since the original content on the server may have been modified. The protocol for verifying cached prefixes was described in detail in Section 5.3.3. The implementation of this protocol using RTSP is described below.

As RTSP is based upon HTTP, the protocol allows for easy validation of cached content. This is performed during the first phase using the `SETUP` request. A client can issue an unconditional or conditional `SETUP` request. An unconditional request is used to request content, whereas a conditional request can be used to verify cached content. If the server determines the content to be valid (through timestamp comparison) then an `OK` response is returned. If the content though has been modified, the response will contain the transport details for the newest version of the content, and the stream will begin once the client sends a subsequent `PLAY` request.

RTSP control messages may either be routed through the overlay network or sent directly to the other party as appropriate. If the message has a known destination (i.e. an IP address) then the message will be sent directly to avoid network delays associated with routing messages through the overlay network.

The initial RTSP `SETUP` request issued by a client to the collaborative cache is treated in a similar manner to a HTTP GET request in Squirrel. This request may be conditional or unconditional, depending on whether the client has a fresh, cached copy of the requested stream prefix. The request is initially directed to the *Home Node* for the content item, based on an identifier derived from the request URI.

The unconditional case where the requesting client does not have a copy of the required prefix in its local cache, is examined first.

In the simplest case, if the *Home Node's* directory does not contain an entry for the requested prefix, or a suitable *Delegate* for the request is not available, the *Home Node* will issue an RTSP `NOT FOUND` response to the client (case #1 in Section 5.3.3).

If however, the *Home Node's* directory contains one or more *Delegates* with a cached copy of the requested content and at least one of the *Delegates* is available to service the request, the request is forwarded to a selected *Delegate.* If the *Delegate's* cached prefix is fresh, it will send an RTSP OK response to the client, establishing a new session (case #6 in Section 5.3.3). If the cached prefix is stale, the *Delegate* will issue a conditional RTSP `SETUP` request to the server, the response to which will indicate whether the cached prefix has been modified. If the prefix has not been modified, the *Delegate* will send an RTSP OK response to the client (case #9 in Section 5.3.3). If, however, the prefix has been modified, a new copy of the prefix must be retrieved from the server. The collaborative prefix cache differs from Squirrel at this point. In Squirrel, a *Delegate* whose cached prefix has been modified will receive an up-to-date copy of the web content from the server, in response to the conditional HTTP `GET` request. Retrieving a media stream that is not required locally, however, is likely to be a prohibitive use of client resources. Accordingly, the *Delegate* issues an RTSP `NOT FOUND` response to the client and informs the directory that the prefix has been modified, causing it to be removed from the directory. The client will retrieve the entire stream from the server, caching the new prefix and eventually becoming the sole new *Delegate* for the stream (case #10 in Section 5.3.3).

The case is now examined where a client has a locally cached prefix but the cached copy is stale. Again, the collaborative prefix cache behaves in a similar manner to Squirrel. The client issues a conditional RTSP `SETUP` request to the *Home Node*. Like Squirrel, the collaborative prefix cache maintains time-to-live information for each cached prefix. If the directory entry is fresh and the client's prefix has not been modified (it is assumed that this can be established through the use of Last-Modified timestamps) then the *Home Node* issues an RTSP `NOT MODIFIED` response to the client (case #4 in Section 5.3.3). If the prefix has been modified, then the request is forwarded to a chosen *Delegate* and handled as described above (case #5 in Section 5.3.3). If the directory entry is not fresh, the client is instructed to contact the server to either validate its existing cached prefix (case #8 in Section 5.3.3) or retrieve the entire stream from the server (case #7 in Section 5.3.3), thereby caching a new prefix. In either case, the client will update the *Home Node* with new information, allowing the *Home Node* to either eventually validate all of the existing *Delegates* or replace them with the requesting client as the new sole *Delegate*.

When a client receives an RTSP OK response from a *Delegate* in response to a `SETUP` request, initially sent to the *Home Node* through the overlay network, both it and the chosen *Delegate* become participants in a new RTSP session. Although it would be possible at this point for the client and *Delegate* to continue to communicate using the peer-to-peer network, a scheme in which the participants transfer to a more efficient, direct TCP connection is proposed. This has the effect of limiting the use of the peer-to-peer network to the location of *Home Nodes* and the occasional exchange of cached prefix metadata between *Delegates* and *Home Nodes*, thereby reducing the latency experienced by clients.

6.2 Server Stream Control Protocol

The Real Time Streaming Protocol (RTSP) can also be used as the control protocol for the multicast enabled server within the Peer Assisted Multicast Streaming architecture.

As was described in Section 3.2.2, an RTSP session can be viewed as having three distinct phases. During the first phase, which begins when a client issues an

RTSP `SETUP` request to the server, the transport mechanism including the multicast address to which the client has been assigned, is agreed. The multimedia stream is delivered in the second phase, in response to `PLAY` and `PAUSE` requests. The third phase is the tear-down phase, which is initiated when either the client or the server issues an RTSP `TEARDOWN` request to end the streaming session.

Clients receiving a multimedia content stream using PAMS typically receive two streams: a prefix stream containing the start of the content from another client, or *Delegate*, participating in the collaborative cache and a second multicast stream providing the remainder of the content from a server. As described in Section 5.4, the server in the proposed architecture will act as a batching server. The server will receive requests from clients that specify the content item that is to be delivered as well as the start offset and duration of the content, or *range*, that should be delivered. If the requesting client can obtain a prefix for the content from the collaborative cache, then it would normally request a range from the server from the end of the prefix to the end of the content. It will also specify the time that it expects to begin receiving the stream. This time is specified as a duration and the stream can begin anytime before that duration has elapsed. To allow for network transmission delays the server will always begin the stream earlier than the time specified by the client. Alternatively, a client that was unable to obtain a prefix for the requested content item from the collaborative cache would request the entire content item, as well as indicating that the stream should be delivered immediately.

RTSP provides the necessary degree of control over both the unicast and multicast streams required by PAMS. When a client issues an RTSP `PLAY` request, it can specify both the range of the content item that should be delivered, as well as the time at which the server should begin sending the stream, using the RTSP `RANGE` header. An example of a `PLAY` request that would be issued by a client that has obtained a prefix from the collaborative cache is shown in Figure 6.2 (a). This request specifies a range beginning at the end of the prefix and continuing to the end of the content item and a start time corresponding to the time the client will receive the end of the prefix stream

RANGE	TIME
Lp - Lt	T + Lp

Figure 6.1: (a) RTSP `PLAY` request issued by a client at time T, for a content item of duration Lt, that has obtained a prefix of duration Lp from the collaborative cache

RANGE	TIME
0 - Lt	-

Figure 6.2: (b) RTSP `PLAY` request issued by a client at time T, that requires the entire content stream of duration Lt from the server

and must begin receiving the remainder of the content. This request can be compared with a `RANGE` header that would be issued by a client that requires the entire content stream from the server, illustrated in Figure 6.2 (b). This request specifies a range corresponding to the entire content item and, since the start time has been omitted, the server should begin transmitting the stream to the client immediately.

The same approach to the implementation of Patching described in Section 3.2.2 may be applied to an implementation of Peer Assisted Multicast Streaming. The primary difference between the protocols required by PAMS and Patching is that, in the case of PAMS, the client will receive just one multicast stream from the server.

As with Patching, the multicast address provided to a client in response to its `SETUP` request in the first phase may be invalidated before the client attempts to enter the second phase with a `PLAY` request as a result of the batching policy implemented by the server as described in Section 5.4. This problem can be resolved for PAMS in the same way by causing the client to re-issue its `SETUP` request if required, as indicated by the server's response to the `PLAY` request.

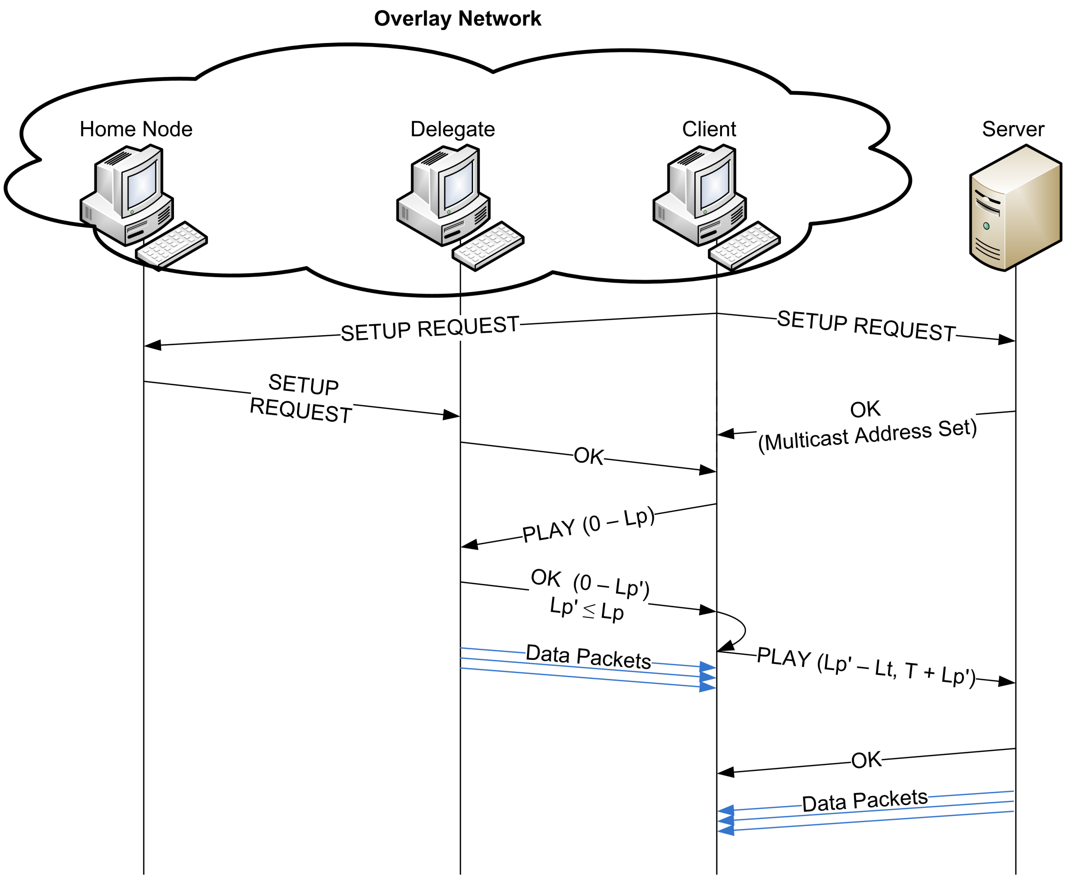


Figure 6.3: PAMS Client Communication

6.3 Client Communication

The preceding sections have described how a client interacts with the collaborative cache and the server using RTSP. Although the two RTSP sessions are separate there is a causal relationship between them. Figure 6.3 illustrates how a client will retrieve the entirety of a content item from the two separate sources. In this example it is assumed that the prefix is available in the collaborative cache and that the multicast address provided in response to the `SETUP` request is still valid when the client issues its `PLAY` request.

6.4 Simulation

A detailed performance analysis of Peer Assisted Multicast Streaming (PAMS) will be described in Chapter 7. Due to the large number of network nodes required to fully evaluate a peer-to-peer based system and the high level of concurrent sessions required to evaluate a multicast based multimedia system a simulation approach is proposed as being more appropriate than a physical evaluation of PAMS.

For the simulation to address all of the issues that may arise in an implementation of PAMS, a highly detailed approach was taken. This approach modelled the activity of PAMS down to the individual network packet level. The simulation was constructed using the OMNeT++ [Var01] discrete event simulation system. Within the simulation servers and clients are modelled as independent entities. At the application layer within these entities, a model of RTSP was developed, upon which on-demand multimedia streaming applications could be constructed. To model the transport and network layers, the IPv6SuiteWithINET [1] framework was incorporated into the model with some modifications to incorporate PIM-SSM. To allow for the modelling of peer-to-peer applications, the OverSim [BHK07] framework was used as it provides a simulation of Chord among other protocols.

6.5 Summary

Following the design of Peer Assisted Multicast Streaming (PAMS) architecture presented in Chapter 5, this chapter has proposed an approach for implementing PAMS using standard network protocols and an existing overlay network protocol. An evaluation of this implementation of PAMS will be presented in Chapter 7.

[1]`http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/IPv6Suite`

Chapter 7

Simulation and Evaluation

Peer Assisted Multicast Streaming relies on a large community of clients participating in a collaborative cache that can store and supply prefixes of content to each other. A small-scale evaluation of PAMS with, for example, tens of clients would be insufficient to clearly and accurately demonstrate the benefit of the approach. Since coordinating a real-world evaluation of PAMS involving thousands of clients would be infeasible, simulation has been selected as the most appropriate way to evaluate the approach.

Simulation also has the advantage of facilitating a more detailed analysis of the behaviour of PAMS than would otherwise be feasible, since a global view of the state of the entire system can be obtained in a simple manner at any time. (For example, the results and analysis in Experiment 8 in this Chapter would have been very difficult to obtain outside of a simulation environment.)

To ensure, as far as possible, that the results obtained through simulation reflect those that would be obtained in practice, a highly detailed discrete event simulation has been implemented, which includes, for example, the simulation of communication protocols from the Application Layer to the Network Layer in the OSI reference model. The simulation experiments range in scale from 1,000 to 8,000 clients participating in a collaborative PAMS cache and requesting content from libraries containing from 1,000 to 20,000 items. The mean number of stream sessions simulated in each experiment ranged from 21,600 to 172,800 streams, with an average of between 900 and 7,200 clients concurrently receiving streams.

This chapter will describe the simulation framework used to evaluate the performance of PAMS and the parameters and metrics used in the experiments. It will then describe the eight separate experiments used to evaluate PAMS, presenting the motivation for each experiment, the parameters used and the results obtained, along with a detailed discussion of each set of results.

7.1 Simulation Framework

Rather than design and develop a new discrete-event network simulation, the OMNeT++ discrete event simulation has been used. OMNeT++ [Var01] is a C++-based, object-orientated discrete event network simulator. OMNeT++ provides a modular framework in which C++ is used to implement the behaviour of *simple modules*, which are then composed into *compound modules* to implement more complex components, such as servers, clients and routers. OMNeT++ has a substantial user base, members of which have contributed complex network models, including models for IPv6 and IPv4 networks. The modular approach allows the use of a level of detail that is appropriate to experimental requirements. For this evaluation, an IPv6 network was modelled with a level of detail that includes accurate modelling of Internet Layer and higher protocols and simplified modelling of lower level protocols. This approach represents a balance between the accuracy of the model and the feasibility of conducting large-scale simulations with thousands of network nodes in a practical time-frame.

As latency is an issue for multimedia streaming applications the simulation must deal with the time taken for packet delivery from the server to the client in order to accurately reflect a real world application. Fully modelling additional network utilistion caused by factors outside of PAMS, would be infeasible due to the increased time to simulate the system. To allow for a smooth handover in the client between the prefix and the shared multicast stream, the server will assume that the latency between itself and the client is one second. This will manifest in the server scheduling the required multicast stream one second before the earliest start time specified in the client request. Based on available statistics describing typical network performance [Ver15] [AT5], the use of a one second overlap between the prefix and multicast streams will be appropriate in most scenarios. However as the average prefix length used in the experiments is six hundred seconds, reducing this by one second will only have a minor impact on the maximum number of concurrent multicast streams required to provide a particular content item from the server.

7.2 Experiment Parameters and Performance Metrics

In each of the experiments described in this Chapter, requests were generated at a mean rate necessary to obtain a desired number of concurrent on-demand streaming sessions. Little's relation [Lit61] may be used to determine the request rate necessary to achieve a desired number of concurrent sessions, for any given session duration. For example, to achieve a mean of 900 concurrent sessions, L, each lasting one hour or 3,600 seconds, W, the required request rate, λ is given by:

$$\lambda = \frac{L}{W} = \frac{900}{3,600} = 0.25 \text{ requests per second}$$

Since the actual number of concurrent sessions will fluctuate and since some clients will be inactive between successive streaming sessions, a client population larger than 900 is required. A default client population of 1,000 has been used in the experiments to evaluate PAMS. These 1,000 clients will all participate in the collaborative cache at all times, regardless of whether they are currently engaged in a streaming session,with the exception of Experiment #8, in which clients will leave and join the collaborative cache over time.

To model the popularity of individual content items, a Zipf distribution [Zip49] has been used. The probability, $P(i)$, of a client requesting the i^{th} most popular content item is given by:

$$P(i) = \frac{1}{i^{\theta}\left(\sum_{j=1}^{N}\frac{1}{j^{\theta}}\right)}$$

where N is the total number of content items and θ is the skew factor. The skew determines what the differences in popularity of various content items will be. For a low skew value, the probability of a request being for a particular item is evenly distributed, whereas, for a high skew value, the probability of a request will be much higher for a subset of content items.

The default prefix duration of 600 seconds the was chosen to allow a large enough batching window so as to demonstrate the benefit of PAMS, but to not be so long so as to overburden the clients participating in the collaborative cache.

Another parameter investigated is the size of the PAMS prefix cache at each client, measured in numbers of prefixes. The default cache size used was large enough to store five prefixes. This default size was chosen to be conservative but increasing the cache would also increase the management overhead of PAMS, especially for the *Home Node* of the most most popular content. Experiment #5 demonstrates the effect of changing the cache size.

The mean interarrival time between successive client requests in the simulation is $\frac{1}{\lambda}$. The actual interarrival times are generated by the simulation using a Poisson distribution with mean λ. When a new request event is generated, a client is selected at random from the set of currently idle clients. The content item requested by the client is determined by the simulation using a Zipf distribution, as described above. The client then establishes its streaming session as described in Chapters 5 and 6.

7.2.1 Independence of Parameters

Each parameter cannot be taken in isolation and must be evaluated in the context of the other parameters. For example the request rate isn't a parameter in itself, it is a product of the mean stream duration and the target mean number of concurrent sessions. In order to properly investigate the benefit of PAMS, both a sufficiently long session and a large number of concurrent sessions is required, thus producing a high request rate for the simulations.

The request rate for each individual content item determines the effectiveness of PAMS. The individual parameters effecting the mean number of concurrent sessions of each content item are the number of content items, the skew factor of the Zipf distribution and the request rate for all items. The number of content items and the skew of the Zipf distribution effect the probability of a request being for a particular content item. For example, by increasing the number of content items the probability of a request being for any of the previous number of content items will decrease. By increasing the skew factor the probability of a request being for one of the more popular titles will increase. These effects are examined in Experiment #4 and Experiment #6.

By increasing the overall request rate, the request rate for each individual content item will increase, therefore increasing the number of concurrent sessions for each content item. The higher the request rate, the better PAMS will perform as demonstrated in Experiment #2.

The duration of the prefix used is one of the most significant parameters effecting the performance of PAMS. The performance of PAMS is highly dependent on the mean number of requests for each content item that arrive during a batching window. Therefore by increasing the prefix duration the more requests will arrive during the batching window. Larger prefixes however, will transfer more of the load onto the individual clients, rather than the server. The impact of the prefix duration is examined in Experiment #3.

In order for PAMS to offer a batching window to the server a prefix must be available in the collaborative cache. This not only means that the content item is stored in a participants cache, but the participant has sufficient network resources to provide the prefix. Each participant is limited to serving one prefix at a time so that the load is better shared around the clients. If the cache hit rate is low then the number of full streams provided by the server will increase, thus increasing the bandwidth utilisation of the server.

The clients' cache size is an important factor in the effectiveness of the cache, and this is examined in Experiment #5. Once the clients' cache size is sufficiently large to hold one popular prefix and the prefix of the stream that is concurrently being received, then the cache hit rate is high enough to demonstrate a reduction in the bandwidth utilisation of the server. Prefix duration has a subtle effect on the effectiveness of the cache. As each client is limited to only serving one request at a time, the longer a prefix is, the longer the time the client will not be available to server other requests, thus increasing the probability of there being no participants being able to serve a requested prefix and thus decreasing the cache hit rate. This effect is discussed further in Experiment #3.

The minimum number of clients in the each simulation must be sufficient to maintain mean target request rate. If there are more clients present than the minimum required, it has the same effect as increasing the cache size of each client, but with the added effect of reducing the possibility that there will be no client without the available upstream bandwidth to provide a prefix. Experiment #8 shows that the effect of this is negligible compared with other factors that can result in cache misses.

The remainder of this chapter describes the eight experiments used to illustrate the behaviour and evaluate the performance of PAMS. The experiments evaluate the effect of varying parameters such as the PAMS prefix duration, request rate and cache size.

7.3 Experiment 1: Performance and Behaviour

The purpose of this experiment is to demonstrate the advantage of Peer Assisted Multicast Streaming over simple unicast streaming, where each request is serviced by a dedicated stream. Peer Assisted Multicast Streaming reduces the overall server bandwidth required to serve incoming client requests. This reduction is achieved in two ways. Firstly, when the prefix of a requested item can be served by the collaborative cache, this will reduce server bandwidth utilisation as the server will only need to supply the remainder of the stream. Secondly, serving prefixes from the collaborative cache allows the server to batch together multiple requests for the same content and service these requests with a single multicast stream.

To demonstrate the advantage of Peer Assisted Multicast Streaming, two different simulations will be compared. The first will simulate the use of PAMS with a ten minute prefix duration. The second will use a zero duration prefix, meaning that the collaborative cache is not used and the entirety of each request will be served by a single server stream. This is equivalent to a unicast approach.

Before discussing the results of this experiment, the expected performance of both of the above simulations will be calculated and compared later with the results of the simulation. This will serve as a validation of the simulation.

The mean number of concurrent server streams when using a zero duration prefix, which is equivalent to a unicast approach, can be estimated using Little's relation [Lit61], which may be re-written here as $S = RLt$. If the server receives one request every four seconds ($R = 0.25$), and the mean stream duration is one hour($Lt = 3600$), then the mean number of concurrent streams, S, will be 900.

The overall performance of PAMS with a ten minute prefix duration cannot be estimated in the same simple manner, as the behaviour and efficiency of each individual client participating in the collaborative cache must be considered. In particular, it would be necessary to estimate the probability not only that a particular prefix is contained in the collaborative cache but also that there is a *Delegate* with available bandwidth to supply the prefix. It is still possible, however, to predict the performance of PAMS when considering only the most popular content items, if it is assumed that prefixes of these content items always be present and there will always be a *Delegate* with sufficient bandwidth available. (Our simulation results support this assumption.)

The mean number of clients receiving content item i from the server is independent of the use of multicast but dependent on the probability, H_i, that i can be served by the collaborative cache. For content item i, if the collaborative cache provides a prefix of duration Lp_i of the total stream duration, Lt_i, the probability that a request is for item i is P_i, the probability that item i can be served by the collaborative cache is H_i and the overall request rate is R, then the mean number of clients receiving content item i from the server is:

$$(H_i \times R \times P_i \times (Lt_i - Lp_i)) + ((1 - H_i) \times R \times P_i \times Lt_i) \tag{7.1}$$

For the most popular content item, it can be assumed that $H_1 = 1$ and therefore the mean number of clients receiving content item i from the server may be simplified to:

$$R \times P_i \times (Lt_i - Lp_i) \tag{7.2}$$

For example, using the default simulation parameters of 5,000 content items and a Zipf skew factor of $\theta = 0.7$, according to Zipf's law the probability that a request will be for the most popular content item ($i = 1$) is:

$$P(1) = \frac{1}{1^{0.7}\left(\sum_{j=1}^{5000} \frac{1}{j^{0.7}}\right)} \approx 0.0249 \tag{7.3}$$

With a request rate, R, of 0.25 requests per second, the server will receive $0.25 \times 0.0249 \approx 0.0062$ requests per second for the most popular item ($i = 1$). Given a prefix duration, Lp_1, of 600 seconds, a total content duration, Lt_1, of $3,600$ seconds, the mean duration of the streams provided by the server will reduce to $3,600 - 600 =$ 3,000 seconds. Thus the mean number of clients receiving the most popular content item from the server will be $0.0062 \times 3,000 = 18.60$.

By batching requests for the same content item into multicast streams, the server can reduce the number of outgoing streams required to service these clients. To estimate the number of multicast streams required to service requests for the most popular content item, it is necessary to first estimate the number of requests that the server can batch into a single multicast stream, given by:

$$1 + (R \times P_i \times Lp_i) \tag{7.4}$$

For example, for the most popular content item ($i = 1$), given a request rate of 0.25 requests per second, within the 600 second batching window (provided by using a prefix duration of 600 seconds), 3.74 requests for the most popular item ($i = 1$) will be received during the batching window, in addition to the request that defined the start of the window. Thus, the server should be able to batch together $1 + 3.74 = 4.74$ requests to be served by a single multicast stream. The mean number of concurrent multicast streams required to serve the most popular content item will now reduce to $18.6/4.74 = 3.94$ streams.

This may be compared with the mean number of streams required by a unicast system to serve the same content. The mean number of concurrent unicast streams for the most popular content item will be $3,600 \times 0.0062 = 22.32$. Thus, the use of PAMS reduces the mean number of concurrent streams required to serve the most popular content item by 82.35%. This reduction arises for two reasons, firstly through the use of multicast by the server, as demonstrated above, and secondly, as a result of the collaborative cache serving a portion of the content, thereby reducing the duration of streams provided by the server. The reduction that arises directly by serving prefixes from the collaborative cache, without using multicast, would be:

$$H_i \times \frac{L_p}{L_t} \tag{7.5}$$

Therefore, for the most popular content item, assuming $H_i = 1$, the reduction that would arise by serving prefixes from the collaborative cache, assuming the values used above, would be 16.67%.

The mean number of server streams required to serve requests for N content items, S, may now be expressed as:

$$S = \sum_{i=1}^{N} \left[\left(\frac{H_i \times R \times P_i \times (Lt_i - Lp_i)}{1 + (R \times P_i \times Lp_i)} \right) + ((1 - H_i) \times R \times Lt_i \times P_i) \right] \tag{7.6}$$

For the most popular content items, the greatest contribution to the reduction in the server resources required to serve the corresponding requests arises from the use of multicast, since the server will be able to batch more requests into each multicast stream. For less popular content, the greatest contribution arises by serving prefixes from the collaborative cache.

The performance of PAMS has been evaluated through simulation using the parameters outlined in Table 7.1, and the results of this simulation will be compared with the estimated performance above.

Parameter	**Value**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	**0, 600**
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed accesses

Table 7.1: Experiment 1 Parameters: Investigating General Behaviour.

Discussion

The graph in Figure 7.1 shows the mean number of concurrent streams serving groups of content items. The content items are ranked according to popularity, that is the items with the highest request frequency, and are plotted in groups of 5 items. The groups are plotted left to right in order of decreasing popularity. The difference between the columns for the same popularity group show the savings achieved by using a 10 minute prefix with Peer Assisted Multicast Streaming over using a zero second prefix which is equivalent to a simple unicast system. For the most popular 5 items, Peer Assisted Multicast Streaming achieves a reduction of approximately 70% in the number

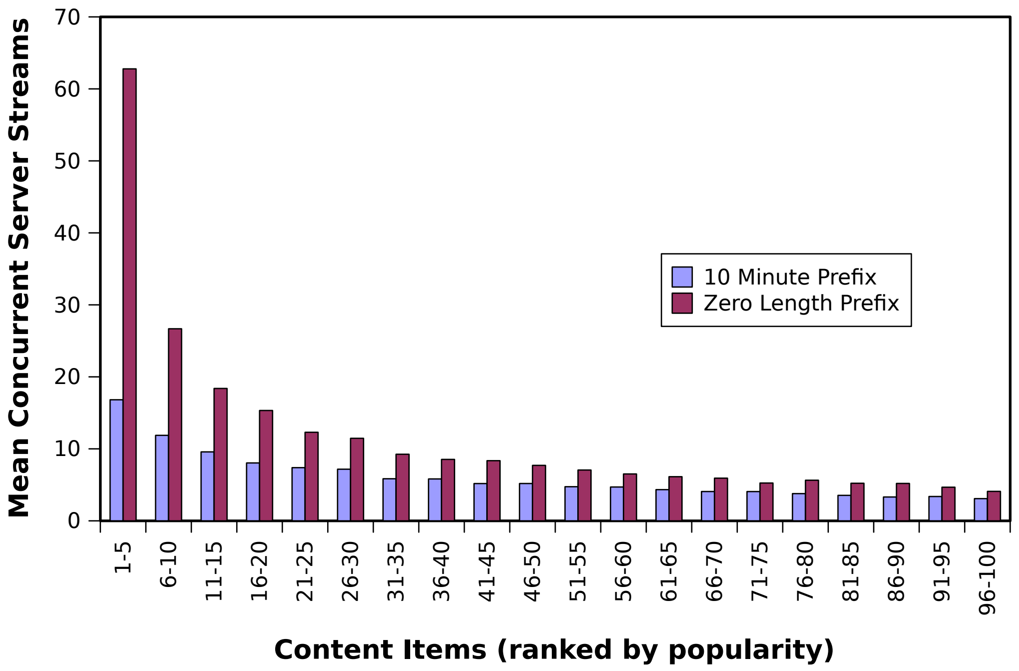


Figure 7.1: Experiment 1 – Comparison of server performance using unicast and PAMS

of concurrent streams. The level of this reduction decreases as the popularity of items decreases, until there is little multicast sharing and the only benefit of the collaborative cache is to serve the prefix.

The simulation results compare favourably with the expected calculated values and thus can be used as a validation of the simulation. For the case of the zero second prefix, the mean concurrent streams served for all content is 898.82 streams, which is close to the mean streams predicted by Little's relation of 900.

Using the Zipf distribution and Little's relation as before, for the simulation of a zero second prefix with a mean stream duration of L_t, a request rate of R and a total number of titles of N, the mean number of streams for the most popular k titles can be calculated as:

$$\sum_{i=1}^{k} \frac{\lambda \times L_t}{i^{0.7}\left(\sum_{j=1}^{N} \frac{1}{j^{0.7}}\right)} \tag{7.7}$$

which, for the parameters listed in Table 7.3, will result in a mean stream count for the five most popular titles of:

$$\sum_{i=1}^{5} \frac{0.25 \times 3600}{i^{0.7}\left(\sum_{j=1}^{5000} \frac{1}{j^{0.7}}\right)} \approx 62.39 \tag{7.8}$$

This, again, is close to the recorded mean value of 62.78 from the simulations as illustrated in Figure 7.1, thus serving as more evidence for the validation of the simulation.

By using PAMS with a 10 minute prefix, the recorded mean number of concurrent streams reduces to 704.48, representing a reduction of 21.62%. If the collaborative cache was operating at maximum efficiency, but the main server was only using unicast streams to provide the remainder of the content, the maximum reduction would be 16.67%, i.e. solely based on serving 600 seconds from the cache and the remaining 3,000 seconds from the server.

The additional reduction from 16% to 21.62% is obtained through sharing of streams, which reduces the mean number of streams for a given title. For the k most popular titles the total mean number of streams serving these titles can be calculated as follows, assuming a 100% cache efficiency:

$$\sum_{i=1}^{k} \frac{R \times P(i) \times (Lt_i - Lp_i)}{1 + (R \times P(i) \times Lp_i)} \tag{7.9}$$

where $P(i)$ is calculated using the Zipf distribution. For the first most popular $k = 5$ titles , a prefix duration of $Lp_i = 600$, a mean stream duration of $Lt_i = 3600$ and a request rate of $R = 0.25$ the equation becomes:

$$\sum_{i=1}^{5} \frac{0.25 \times P(i) \times (3600 - 600)}{1 + (0.25 \times P(i) \times 600)} \approx 16.27 \tag{7.10}$$

The recorded mean stream count for the five most popular titles was 16.82, which again compares favourably with the above estimated result.

Figure 7.2 shows the mean number of requests served per multicast stream. The number of requests served decreases with the decreasing content popularity. This is due to fewer requests arriving for the less popular content during the batching window

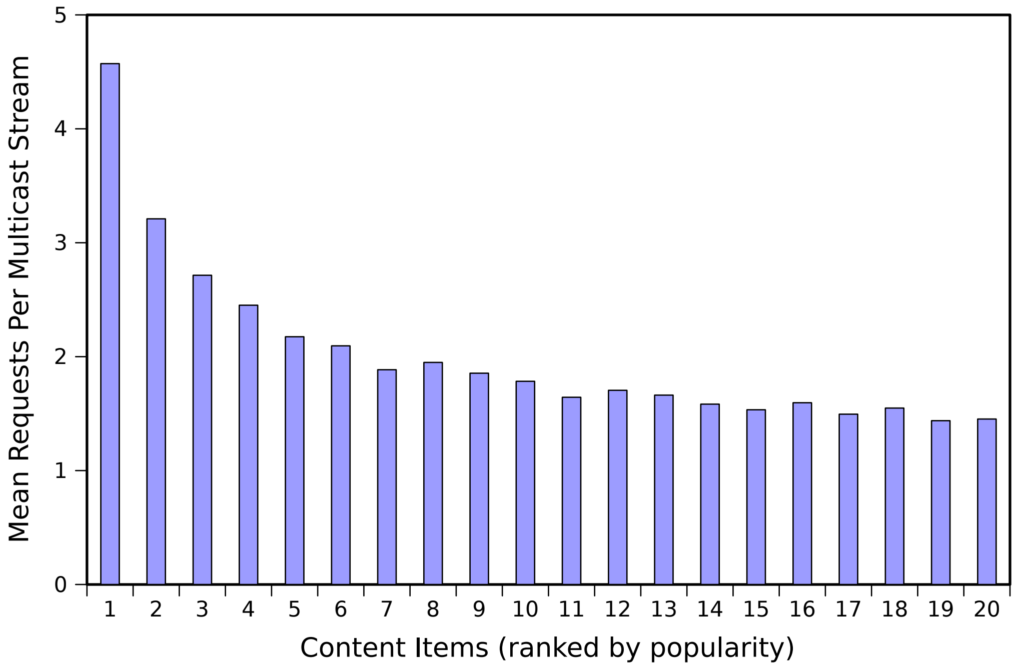


Figure 7.2: Experiment 1 – Mean requests served per multicast stream using PAMS

provided to the server. As discussed previously, the theoretical number of requests served per multicast stream for the most popular content item is 4.74. From the simulation results a mean value of 4.57 was recorded, which again is comparable and serves to validate the simulation.

7.4 Experiment 2: Client Population Size

Increasing the client population size will increase the aggregate size of the collaborative cache. Such an increase, however, will also cause an increase in the overall request rate, assuming each individual client continues to submit requests at the original rate. The purpose of this experiment is to determine what effect such changes in client population size will have on the performance of Peer Assisted Multicast Streaming. The performance of PAMS will be evaluated by measuring the mean number of concurrent streams delivered by the server.

As the overall request rate increases with increasing client population, the mean number of concurrent streams of the most popular content item ($i = 1$) will approach a

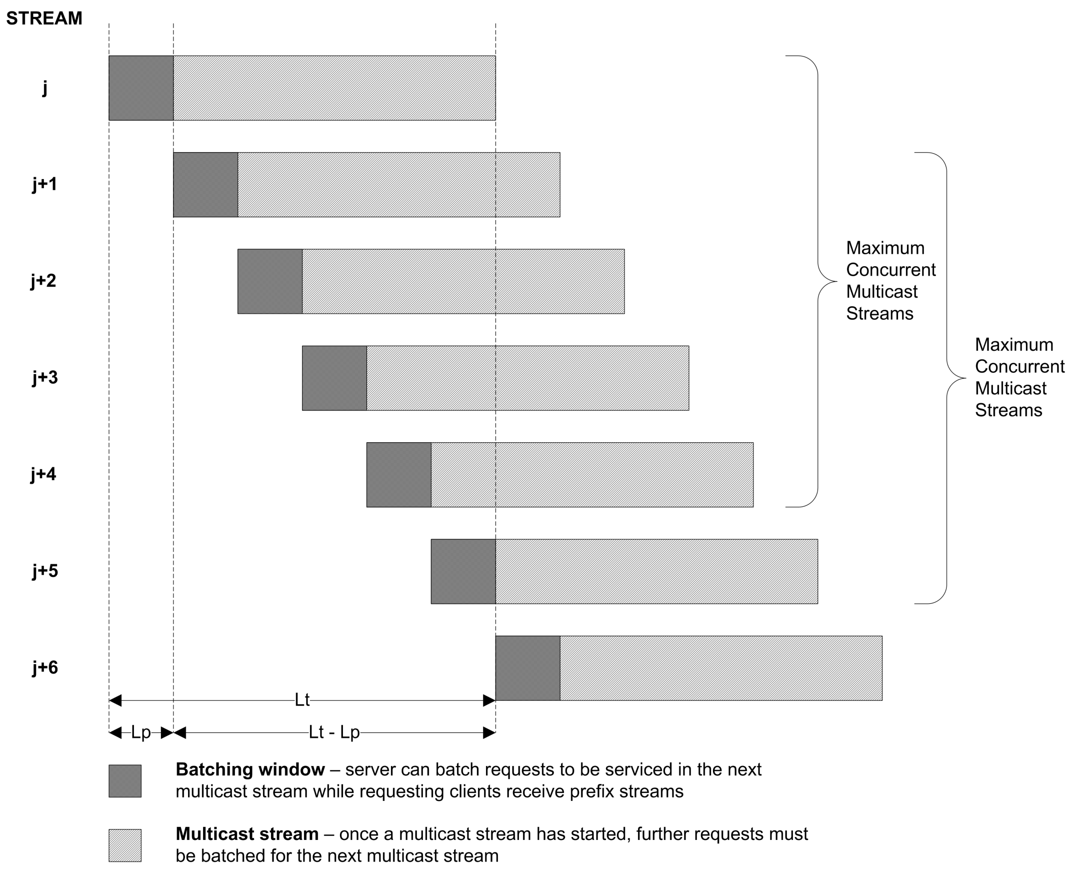

STREAM
j
j+1
j+2
j+3
j+4
j+5
j+6
Maximum Concurrent Multicast Streams
Maximum Concurrent Multicast Streams
Lt
Lp
Lt - Lp
Batching window – server can batch requests to be serviced in the next multicast stream while requesting clients receive prefix streams
Multicast stream – once a multicast stream has started, further requests must be batched for the next multicast stream

Figure 7.3: Maximum Number of Concurrent Multicast Streams Using PAMS

maximum, $Smax_i$. This maximum is determined by the mean stream duration of that content item, Lt_i, and the prefix duration provided by the collaborative cache, Lp_i. For a particular content item, the server only needs to start a new multicast stream for that content at most every L_p seconds, as shown in Figure 7.3. As a result, the maximum number of concurrent streams of that content item is given by the following formula, assuming a 100% collaborative cache hit rate for that content item.

$$Smax_i = \frac{Lt_i - Lp_i}{Lp_i} \tag{7.11}$$

The parameters for this experiment are listed in Table 7.4. The effect of changing client population sizes has been evaluated for populations from 1,000 clients to 8,000 clients in increments of 1,000. This results in request rates ranging from 0.25 requests per second to 2.00 requests per second. All other parameters remain unchanged from the defaults used in Experiment #1.

Parameter	**Value**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	600
Cache Size (number of prefixes)	5
Network Size (number of clients)	**1000, 2000, . . . 8000**
Request Rate (requests/second)	**0.25, 0.50, . . . 2**
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed accesses

Table 7.2: Experiment 2 Parameters: Effect of Client Population Size

Discussion

The results of this experiment can be seen in Figure 7.4. Due to the ability of the system to batch together multiple requests, the mean number of concurrent streams for the most popular content items reaches a maximum of 5.16, which compares to the 5.00 streams predicted by equation 7.11.

This result means that when using PAMS, the maximum resources required to serve popular content could be determined in advance, and it will be independent of the request rate. This is an important result as it means the the loads placed on the main server for serving popular content is highly predictable, especially as the popular content will likely be available in the collaborative cache, thus allowing the server to batch together multiple requests.

The reduction provided by PAMS increases as the population size increases. As the associated request rate increases, the number of requests served per multicast stream is increased for less popular titles thus reducing the server bandwidth required to serve all content. The measured mean number of concurrent streams required to serve all content by PAMS, the theoretical number of concurrent streams required by a unicast system or the theoretical number of concurrent streams required by a unicast system if used with a collaborative cache, are presented in Figure 7.5.

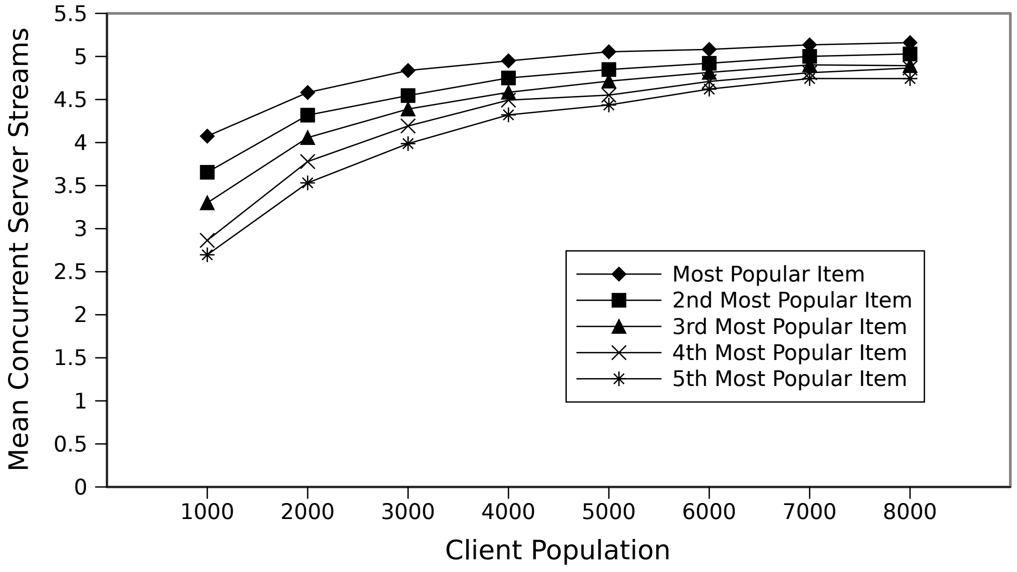


Figure 7.4: Experiment 2 – Effect of population size on the mean number of per title concurrent streams

7.5 Experiment 3: Prefix Duration

The duration of the prefix provided by the collaborative cache will determine the duration of the batching window during which the server can group together multiple requests for the same content into a single multicast stream. The mean number of concurrent server streams can be reduced if more requests can be batched together into a single multicast stream. Furthermore, supplying longer prefixes from the collaborative cache will reduce the remaining duration of the stream that must be supplied by the server, further reducing the number of concurrent server streams. However, increasing the prefix duration will also result in a corresponding increase in the mean number of concurrent streams required from the collaborative cache. This trade-off will be examined in this experiment.

The effect of varying the prefix duration has been evaluated by measuring the mean number of concurrent streams delivered by the server and the mean concurrent streams being delivered by the average client for a range of prefix durations. The remaining parameters are the same as the default parameters used in other experiments. Prefixes ranging from zero seconds (no prefix) to 1,800 seconds have been evaluated.

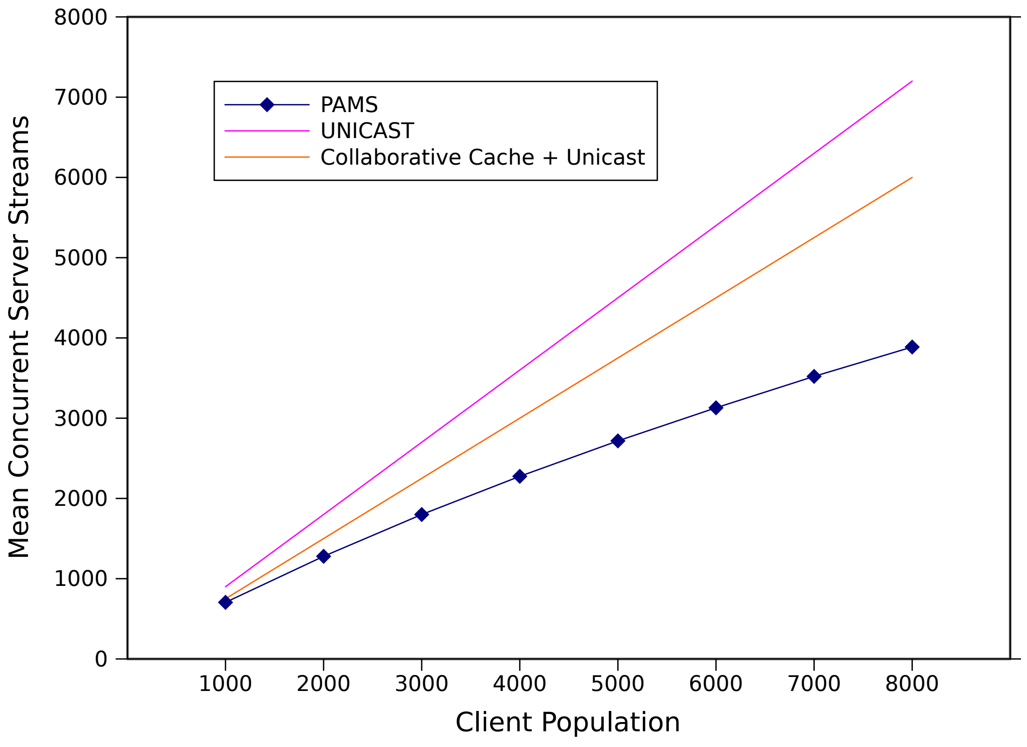


Figure 7.5: Experiment 2 – Effect of population size on server performance

With a zero second prefix the collaborative cache will not be used and, as a result, the performance will be comparable to that of a unicast system. The parameters for this experiment are listed in Table 7.5.

Parameter	**Value**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	**0, 100, 200 . . . 1800**
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed

Table 7.3: Experiment 3 Parameters: Effect of Prefix Duration

Discussion

As the prefix duration increases, the number of concurrent server streams decreases. This is shown by the graph in Figure 7.6.

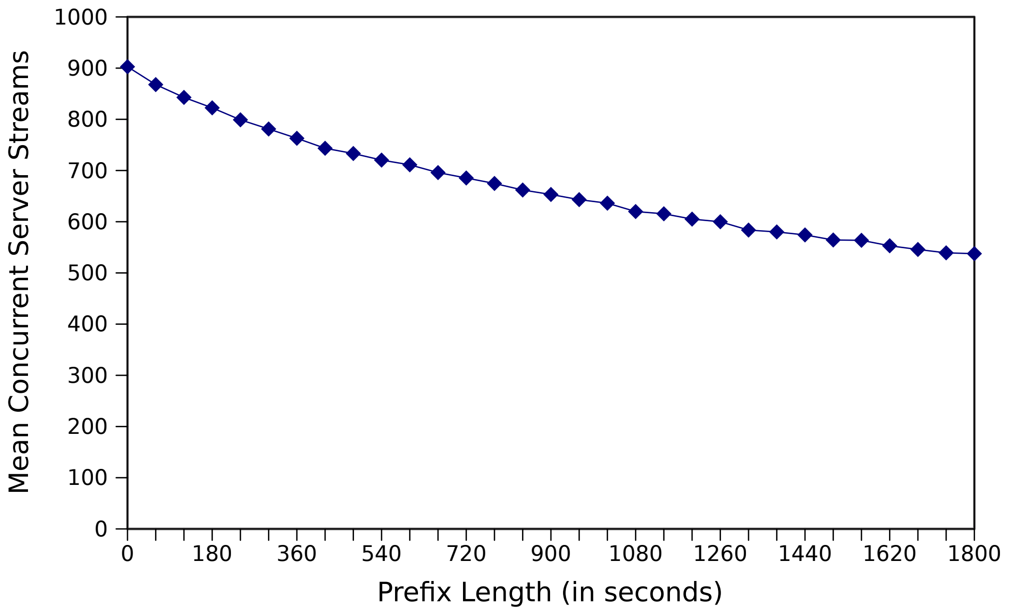


Figure 7.6: Experiment 3 – Effect of Prefix Duration on Server Performance

The decrease in the number of concurrent server streams is non-linearly related to the increase in prefix duration. As the prefix duration increases, it becomes more important that the distributed cache can provide the prefixes requested of it, including the less popular content. As the prefix duration increases though, there will be a corresponding increase in the number of clients concurrently providing prefix streams as can be observed in Figure 7.7. This results in a reduction in the performance of the collaborative cache as the duration of the prefix served increases. As the clients are restricted to serving only one concurrent stream, and it is more likely that for longer prefixes that a client cannot serve the requests directed at it. For example, at a prefix duration of 600s, the collaborative cache can provide a prefix 68.44% of the time, and this results in a reduction of 21.22% of the concurrent streams served by the server when compared with a zero second prefix. When the prefix is increased to 1200s the cache can now only provide a prefix for 65.12% of the requests, and this results in a reduction of 32.97%, which is a lower reduction than would be expected for this prefix duration.

The effect on the client resources is linear, as can be seen in Figure 7.7. As the prefix duration increases, it is more likely that a client will be serving a stream. Regardless of this though, even for a small prefix duration, the overall load on the clients is small for the reduction in bandwidth requirements for the server. If the prefix duration is set at 600 seconds, then the mean number of concurrent streams supplied by each client is 0.09, or of the 1,000 clients in the collaborative cache, a mean of only 90 clients are providing a prefix at any time.

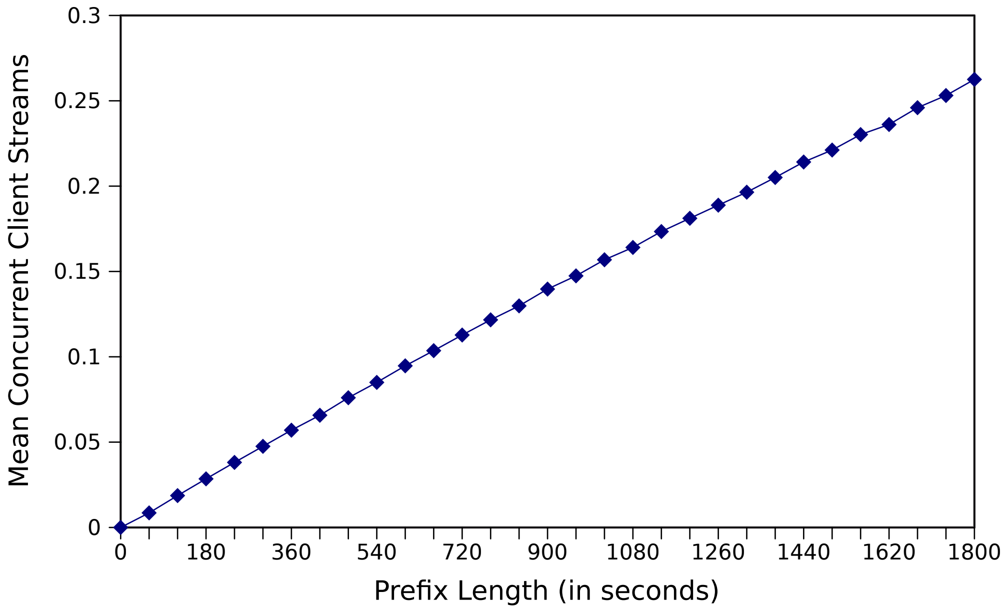


Figure 7.7: Experiment 3 – Effect on prefix duration on client performance (averaged across all clients)

As the prefix duration increases, so too does the batching window available to the server to group together requests. This behaviour can be seen in Figure 7.8. The mean number of requests served per multicast stream increases in an approximately linear manner, with the relative increase greater for the most popular content as defined by Equation 7.4. This increase in number of requests served per stream results in the decrease seen for the overall concurrent streams required by the server.

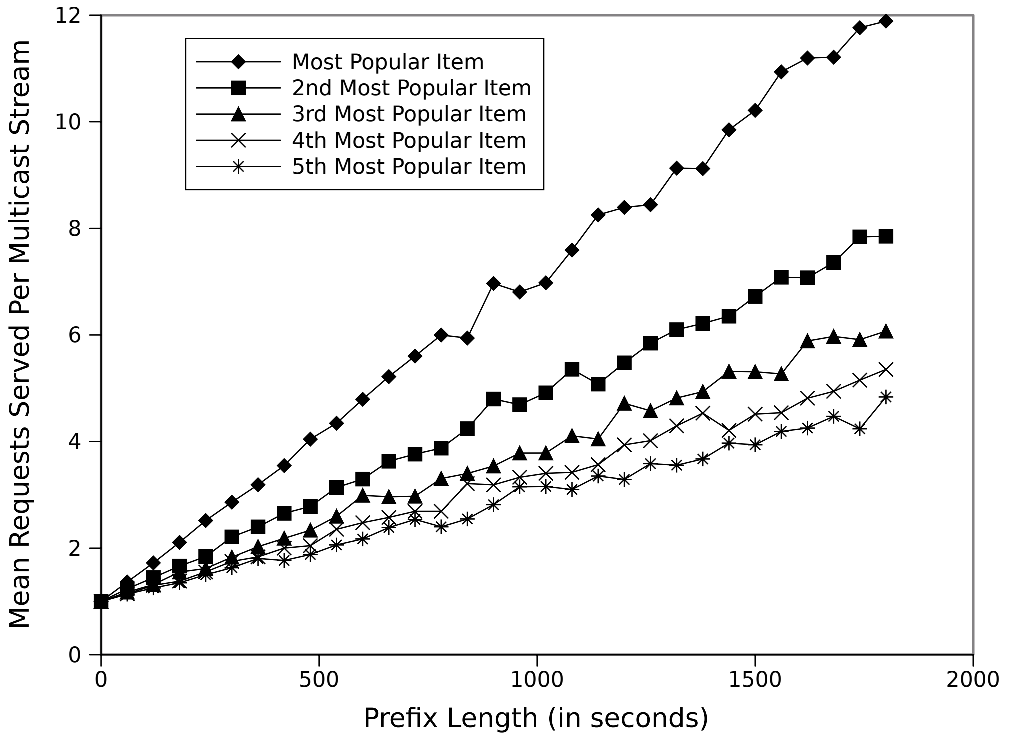


Figure 7.8: Experiment 3 – Effect of prefix duration on requests served per multicast stream

7.6 Experiment 4: Popularity of Content

As has been seen in prior experiments, the popularity of content has a major bearing on the effectiveness of PAMS. Within these experiments a Zipf distribution is used to model the popularity of content. By changing the skew (θ) the probability of a request being for a particular title will change. As θ increases, so will the likelihood of a request being for the first title in the distribution, thus making this title the more popular. For this experiment, θ will range between 0.0 and 1.0. At a skew of 0.0, all titles will have the same popularity, which will result in a reduction of the likelihood of multicast sharing for content, as very few requests for the same content will be received during the batching window allowed by the collaborative cache. Also the more distributed the requests are, the lower the efficiency of the collaborative cache will be. The parameters for this experiment are listed in Table 7.6. The mean number of concurrent server streams, the mean number of requests served per multicast stream and the number of concurrent server streams for popular items will be measured.

Parameter	**Value**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	600
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	**0, 0.1 . . . 1.0**
Local Cache Replacement	Least Frequently Accessed

Table 7.4: Experiment 4 Parameters: Effect of Content Popularity

Discussion

The effect of changing the popularity of content is seen in Figure 7.9. As the requests become more concentrated around a small group of content items, the number of concurrent streams delivered by the server decreases. This decrease is accounted for by both the increased performance of the collaborative cache and the increased ability of the server to group together multiple requests.

As θ increases, both the local and collaborative caches perform better, with the local cache hit rate increasing from 0.09% to 9.50%, and the collaborative cache hit rate increasing from 61.90% to 76.63% over the range of the parameters.

When the requests are more evenly distributed, with lower values of θ, the local and collaborative caches available to a client have a lower chance of providing the prefix for a requested content item, resulting in a longer mean stream duration for the server and fewer opportunities to batch multiple requests into a single multicast stream .

The effect of the popularity skew on the number of requests served per multicast stream is illustrated in Figure 7.10. As θ increases, more of the requests will be for a smaller portion of the most popular titles.

Coupled with improved cache performance,this means the server the have more opportunities to delay the start of multicast streams and will receive more requests during this delay period, allowing more requests to be serviced by each multicast stream.

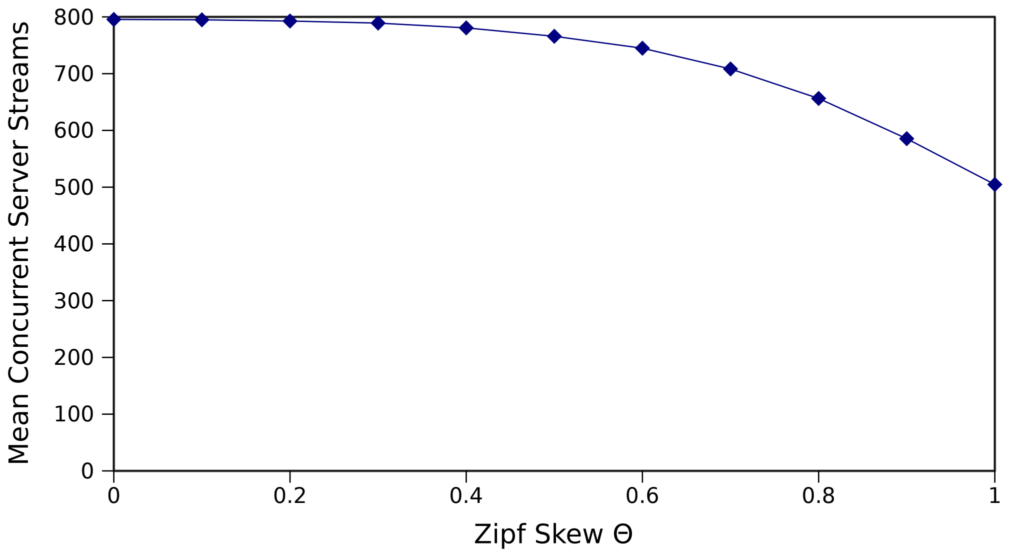


Figure 7.9: Experiment 4– Effect of popularity on server performance

The effect that varying the popularity skew has on the number of concurrent streams for individual content items is shown in Figure 7.11 for the five most popular content items. As more requests arrive for a smaller subset of the available content items, initially the number of concurrent streams will increase for these content items. This increase, is less than the increase in the number of requests received for these content items, so results in an overall reduction in the number of concurrent server streams. As θ approaches 1.0, the number of concurrent streams for the most popular item nears 5 streams, which is the maximum theoretical number of concurrent streams for PAMS using a 600 second prefix.

7.7 Experiment 5: Client Cache Size

While the duration of prefix is a key contributing factor to the effectiveness of PAMS, the collaborative cache's ability to deliver the prefixes can impact significantly on this effectiveness. By limiting the number of prefixes each client stores and makes available to the the collaborative cache, there is an expectation that the performance of PAMS would be significantly affected.

Prefixes are considered to be atomic once fully stored in the cache and, if a prefix

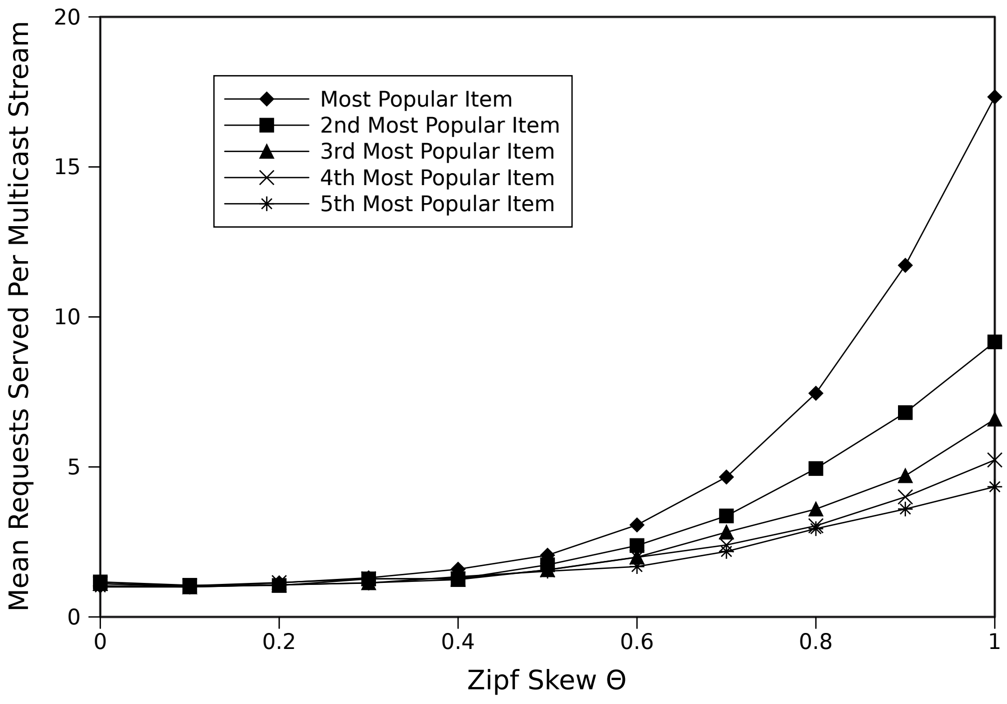


Figure 7.10: Experiment 4 – Effect of popularity on mean requests served per multicast stream

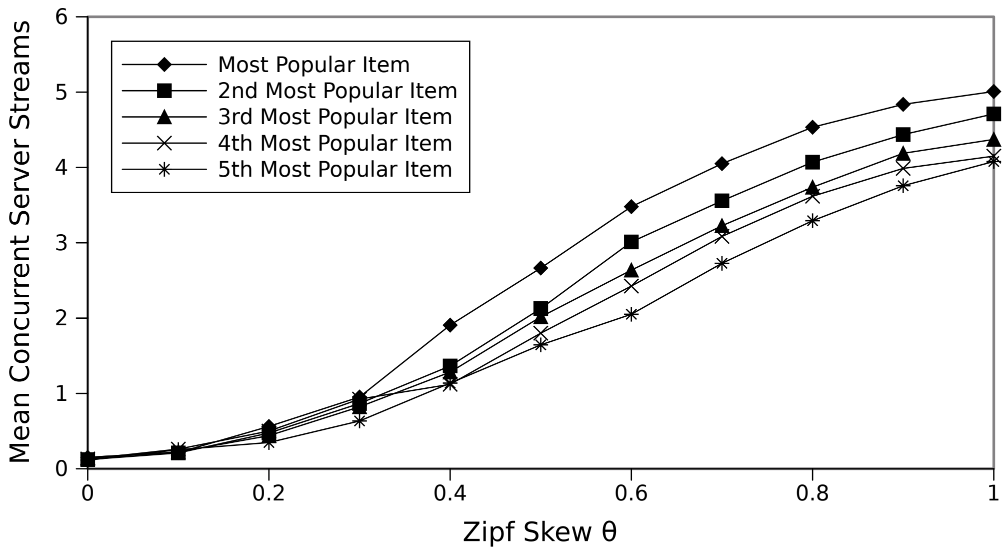


Figure 7.11: Experiment 4 – Experiment 4 – Effect of popularity on server performance for the five most popular items

is being removed, the entirety of the prefix is removed to make space for a new prefix. If the client is replacing a prefix, it will first try to select a prefix that isn't currently being streamed to another client. If no such prefixes exist, it will have to teardown the streaming session with the receiving client. This receiving client, will now need to retrieve the missed portion of the prefix from the main server, instead of from the collaborative cache. As clients are limited to serving only one client at a time, once the cache size is above two, the client will always be able to select a prefix to remove without terminating a streaming session. When the cache size is less then two, though, clients receiving content from the collaborative cache will not always be guaranteed to receive the entirety of the content from the cache once streaming has begun. When the cache size is limited to one prefix, and if a high proportion of the clients participating in the collaborative cache are currently streaming content, then generally the only content available to the collaborative cache is the content that each client is currently receiving.

Parameter	**Value**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	600
Cache Size (number of prefixes)	**0, 1, 2, 3...20**
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed

Table 7.5: Experiment 5 Parameters: Effect of Client Cache Size

Discussion

The effect of the clients' cache size on the performance of the server is shown in Figure 7.12. As the cache size increases from zero, the initial reduction in concurrent streams is large as both the use of multicast and the collaborative cache allows the server to deliver fewer streams. After a cache size of two, the rate of this reduction is decreases as the cache size increases, as the majority of prefixes can now be served from the collaborative cache. For a system based on PAMS to be effective, a cache size large enough to store at least two prefixes must be used, otherwise there is not enough content in each cache to provide enough prefixes to allow for savings based on PAMS.

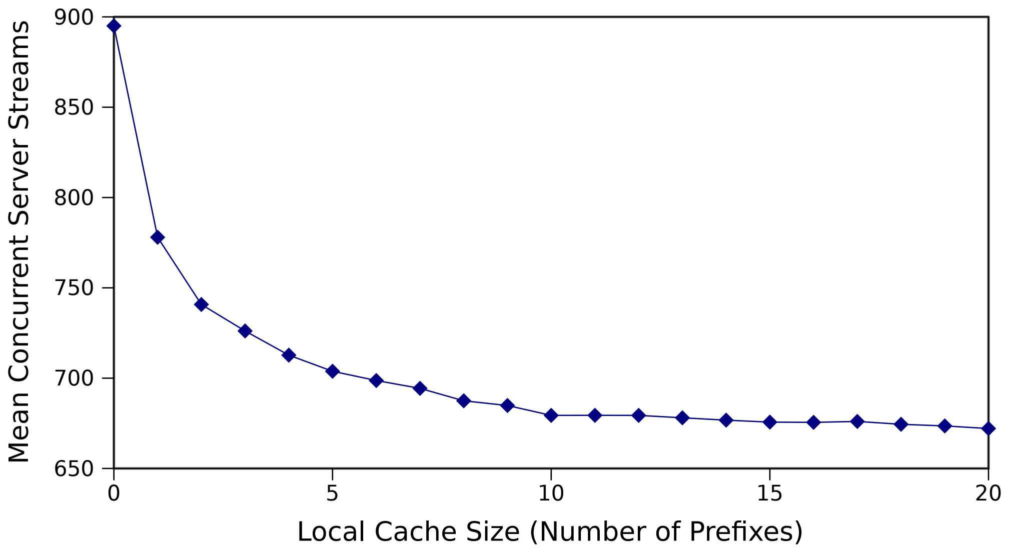


Figure 7.12: Experiment 5 – Effect of cache size on server performance

As the cache size increases, the hit rates of both the local caches and the collaborative cache improve. The effect on the collaborative cache improves dramatically, initially while the local cache improves in a linear fashion and at a more modest rate . The effect that increasing the cache size has on the respective hit rates is shown in Figure 7.13.

7.8 Experiment 6: Number of Content Items

The total number of content items available within an architecture based on PAMS has a sizable effect on performance. As the Zipf distribution has been used to model the popularity of individual content items, as the number of content items increases, the incoming requests will become less concentrated on the most popular items. This will reduce the performance of the collaborative cache and reduce the potential for the server to serve more than one request with each multicast stream.

The secondary effect of increasing the number of available content items is on the collaborative cache. By increasing the number of titles, eventually there will be more titles available from the server than the collaborative cache has the capacity to store, resulting in an increase in the number of requests that must be serviced entirely by the

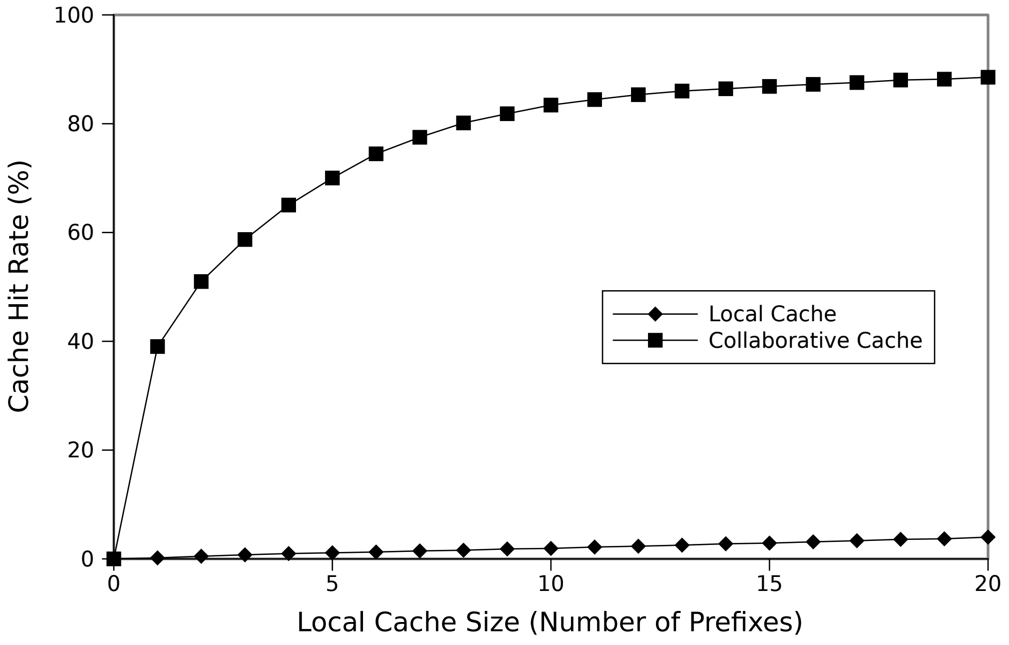


Figure 7.13: Experiment 5 – Effect of cache size on cache hit rate

server.

The effect of varying the number of content items has been evaluated by measuring the mean number of concurrent streams delivered by the server and the average hit rate for the collaborative cache. The remaining parameters are the same as the default parameters used in other experiments. The number of content items simulated ranged from 1,000 to 20,000 for this experiment. The parameters for this experiment are listed in Table 7.8.

Parameter	**Value**
Number of Content Items	**1000, 2000 . . . 20000**
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	600
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed

Table 7.6: Experiment 6 Parameters: Effect of Size of Content Library

Discussion

As stated above, as the number of content items increases, the number of clients requesting the k most popular titles will decrease, as the requests become more distributed. For example, with 5,000 content items, the probability of requesting one of the five most popular items will be 0.069. However, when the number of content items increases to 20,000, the probability of requesting one of the five most popular content items reduces to 0.045, a reduction of 35%. The overall effect of this on the server is demonstrated in Figure 7.14.

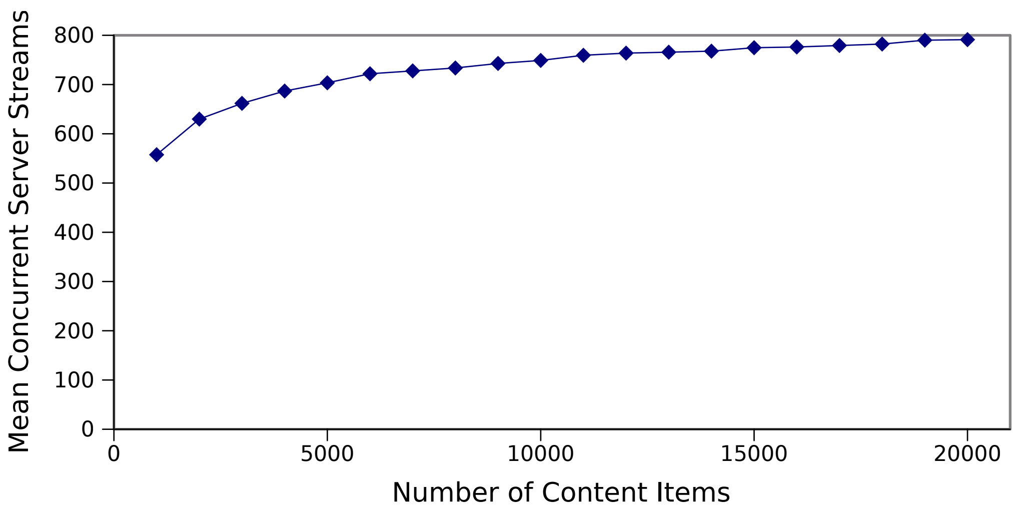


Figure 7.14: Experiment 6 – Effect of number of content items on server performance

As the number of content items increases, the server is required to provide more concurrent streams to service the incoming clients requests. This increase in the number of concurrent server streams occurs for two closely related reasons. Firstly, the performance of the cache decreases, reducing the number of opportunities the server has to batch multiple requests into a single multicast stream. Secondly, when the server does have a an opportunity to delay the servicing of a request during which requests can be batched together, the number of requests that will arrive during the window will be reduced.

Figure 7.15 shows how the number of client requests serviced per multicast stream reduces with an increasing number of content items. The effect on the performance of

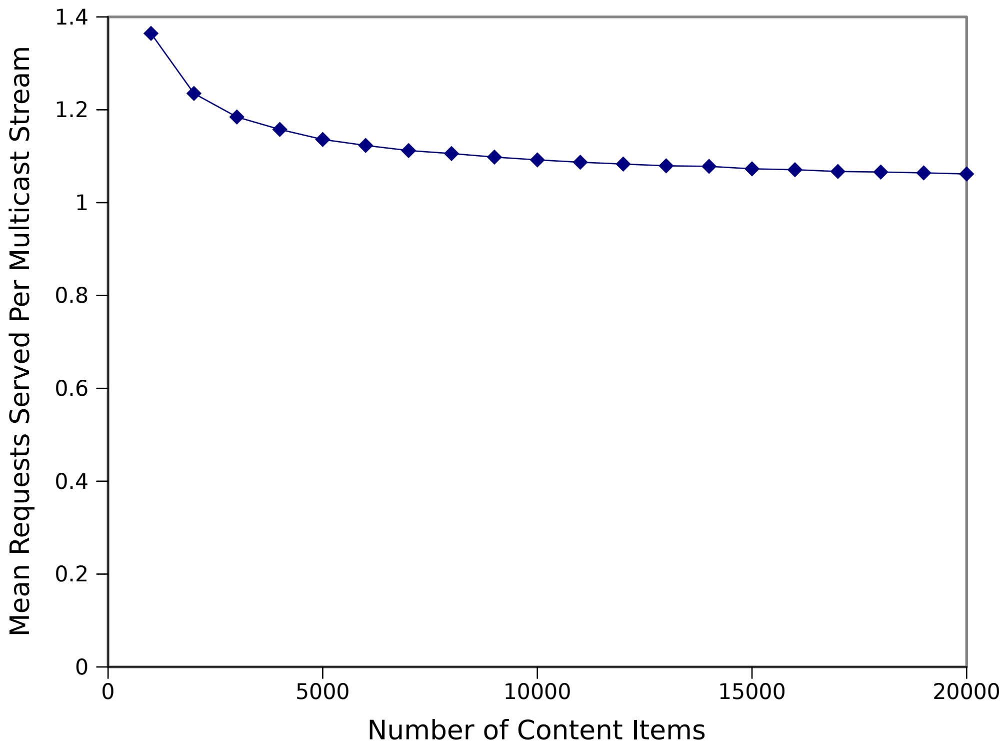


Figure 7.15: Experiment 6 – Effect of number of content items on requests served per multicast stream

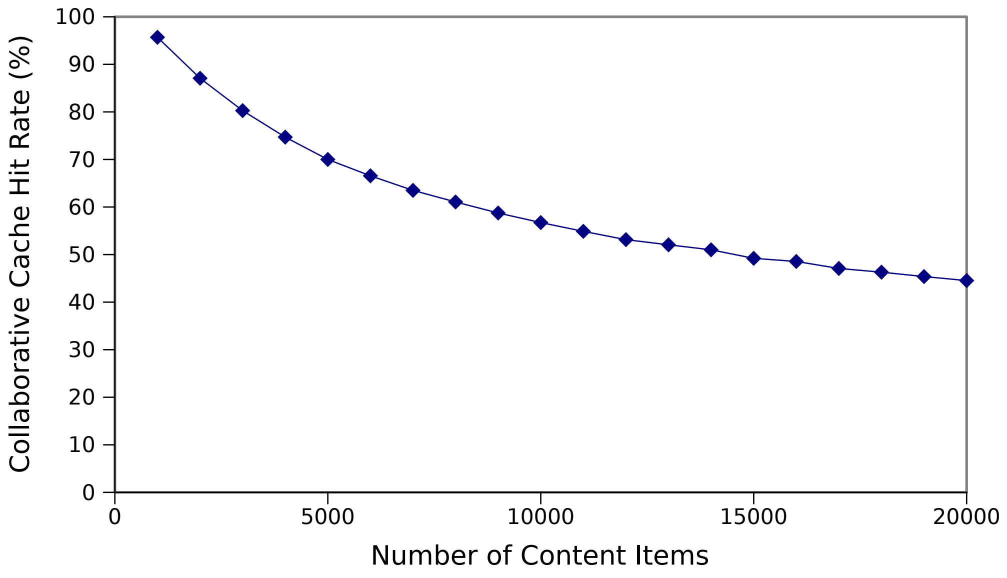


Figure 7.16: Experiment 6 – Effect of number of content items on requests on collaborative cache performance

the collaborative cache is shown in Figure 7.16. As can be seen, with 20,000 content items, the collaborative cache can only provide a prefix to 44.53% of requests, with the vast majority of the cache misses, 98.02%, due to the content not being stored in the collaborative cache.

7.9 Experiment 7: Flash Crowds

All of the experiments to evaluate the performance of PAMS so far have assumed that the set of content items available to clients, and the popularity of those items, remains fixed for the duration of the experiment. This allows the collaborative cache to become populated with enough *Delegates* to serve prefixes of the most popular content items. This will ensure that PAMS can significantly improve the performance of the service.

As it is not a realistic assumption that the popularity of content will remain unchanged or that new content will not be introduced. The effectiveness of PAMS will now be evaluated in situations where new highly popular content is introduced. A significant increase in the expected popularity of a content item, or a significant increase in the overall request rate is commonly referred to as a “Flash Crowd” [JKR02].

To model a flash crowd in PAMS, a new title is introduced, and the likelihood of a request being for this item is again based on a Zipf distribution. This new item becomes the most popular item in the Zipf distribution, and all other items shift down in popularity. As the item is new, there will be no instances of the prefix contained in the collaborative cache, resulting in the server being initially unable to batch together requests for this item. Until the cache can provide enough prefixes to match the request rate, the performance of PAMS will be reduced. The extent of this reduction will be evaluated in this experiment.

To measure the performance of PAMS while stabilising after the introduction of new content, the number of concurrent streams required to serve all requests for the new content will be measured over time.

The timing of content announcement of new *Delegates* for a content item and the selection of *Delegates* for incoming requests becomes important for reaching stability quickly and efficiently. PAMS will be evaluated with two different content announcement policies. With the first policy, which has been called *Complete Prefix*, a *Delegate* will not announce the availability of a prefix until the entire prefix has been stored in its local cache. This allows for clients requesting content from the collaborative cache to be certain of the duration of the prefix they will receive. For the second policy, *Incomplete Prefix Delegates* will announce availability of prefixes once the start of the prefix has been received. Any requests directed at these *Delegates* will result in only some of the prefix being served, with the remainder required from the server along with the remainder of the content stream. The directory will prioritise selecting complete prefixes over incomplete prefixes, so that generally, in a stable system, only *Delegates* with complete prefixes will be selected. This selection policy has been previously described in Section 5.3.2.

The policy for selecting which incomplete prefix to use is also an important consideration. The longer a prefix that can be provided by the collaborative cache, the more likely it will be that the server can group together multiple requests for the same content item. To achieve this, instead of fairly selecting a *Delegate* based on a round robin system, if no complete *Delegates* are available, the *Home Node* will select the *Delegate* with the longest incomplete prefix. While the *Home Node* will not know exactly how long the prefix is, it will assume that the *Delegate* which has been present in the directory the longest has the longest available prefix. Selecting the longest incomplete prefix will be evaluated in comparison to selecting incomplete *Delegates* based on round robin selection. These two policies result in the further classification of the *Incomplete Prefix* policy as *Incomplete Prefix with Round Robin Selection* and *Incomplete Prefix with Longest Prefix Selection.* The parameters for this experiment are shown in Table 7.9.

Parameter	**Value**
Delegate Selection Policy	**Complete Prefix** **Incomplete Prefix with Round Robin Selection** **Incomplete Prefix with Longest Prefix Selection**
Number of Content Items Introduced	1
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	**600, 1800**
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed

Table 7.7: Experiment 7 Parameters: Impact of Flash Crowds

Discussion

The graph in Figure 7.17 shows the effect that introducing new content has on the server. The x-axis shows time, and the y-axis shows the number of concurrent streams required by the server to serve all requests for the new content item. The new content item is introduced at time 0 on the graph. The three PAMS policies: *Complete Prefix*, *Incomplete Prefix with Round Robin Selection* and *Incomplete Prefix with Longest Selection* are shown on the graph. These PAMS policies are compared with a unicast system, again with unicast being represented by a simulation using a zero second prefix.

When the new content item is introduced at time 0 requests for this content item begin to be received by the server. For all three PAMS policies, the first request must be served in its entirety by an individual server stream that cannot be used to service any other requests as there is no prefix available in the cache. When the *Complete Prefix* policy is used, the number of concurrent streams required by the server increases steadily. This is because all requests require the complete stream from the server, until the first client to request the new content becomes a *Delegate* and can provide the prefix to other clients. Even when the first client has become a *Delegate* for the prefix, it can only serve this prefix to one client at a time, so the number of streams required

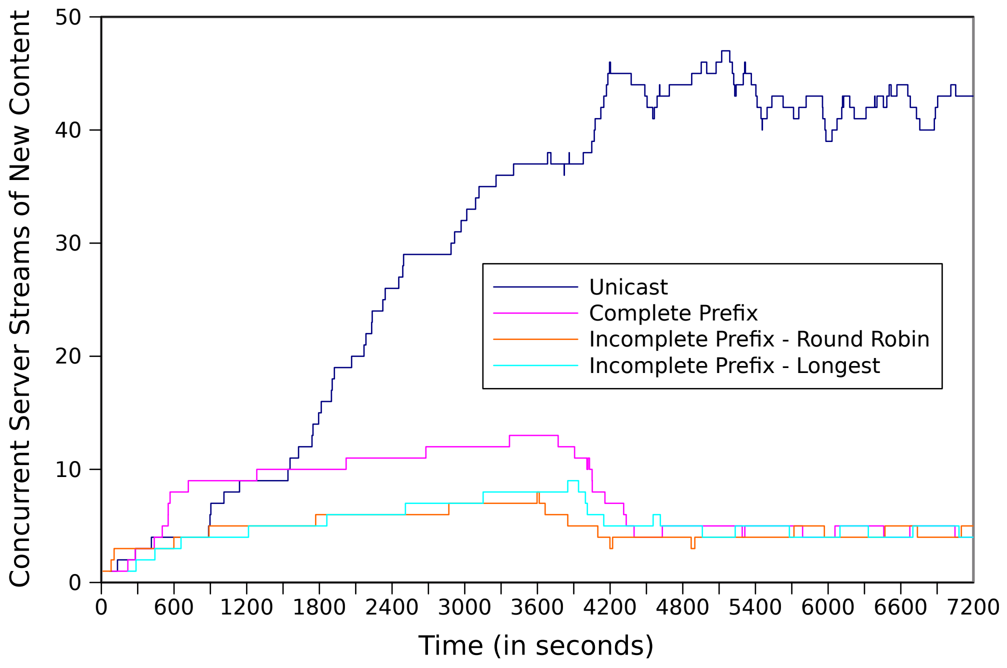


Figure 7.17: Experiment 7 – Effect of a flash crowd on server performance (600s Prefix)

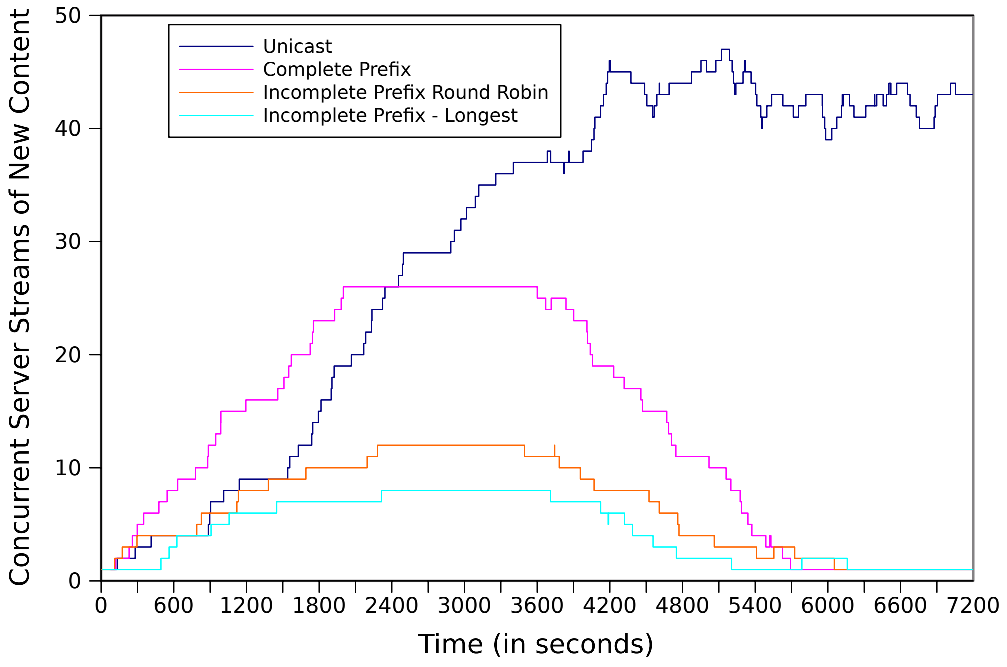


Figure 7.18: Experiment 7 – Effect of a flash crowd on server performance (1800s Prefix

by the server will continue to rise. For a prefix duration of 600 seconds, the number of concurrent server streams required when using the *Complete Prefix* policy peaks at thirteen concurrent streams, before reducing as the system stabilizes.

Alternatively, when using one of the two variants of the *Incomplete Prefix* policy, the peak in the number of concurrent server streams caused by the introduction of the new popular content item is significantly reduced. This is due to the server being allowed a small but increasing window in which to batch together requests, resulting in fewer concurrent streams being required to serve all of the requests for the same new popular content item.

The difference between the *Incomplete Prefix with Round Robin Selection* and *Incomplete Prefix with Longest Selection* policies is clearer when a longer prefix is used, as seen in Figure 7.18. This graph shows the effect of new content being introduced in a system using prefixes which are 1,800 seconds long. With longer prefixes the peak number of concurrent server streams is reduced when using the longest available prefix as the duration of the batching window made available to the server will be maximised.

7.10 Experiment 8: Churn

Clients participating in a collaborative cache will join and leave the cache over time. This type of behaviour can be due to the normal pattern of usage of a system, when a client joins the system to view some content, or through unanticipated events such as a machine powering down unexpectedly. It should be noted, however, that this behaviour will depend on the implementation. For example, if PAMS was to be implemented by set-top boxes, then the set-top boxes could reasonably be expected to participate in the collaborative cache even when they are not being actively used by their owners. Alternatively, if clients correspond to laptop computers, connectivity to the collaborative cache may be expected to be more intermittent.

This type of client behaviour within a peer to peer system is commonly referred to as “churn” [RGRK04]. Churn can destabilise overlay networks as key routing information can be lost when a peer leaves the network. Applications built on top of overlay networks can also be affected, especially if only a small number of peers hold information that is vital to the operation of the application.

Churn will reduce the performance of the PAMS collaborative cache in a number of ways. Firstly, and most significantly, through the loss of *Delegate* information for the most popular content items. Secondly, through destabilisation of the underlying overlay network used to route lookup requests through the collaborative cache. Finally, but less significantly, churn will result in the removal of *Delegates* for prefixes as those *Delegates* leave the collaborative cache.

Delegate information for one content item is only stored by the *Home Node* for that content item. If this node were to leave the collaborative cache, another node would take over as *Home Node*, but the new *Home Node* would be unaware of any existing *Delegates* for the content, meaning that requests for prefixes would be rejected until one or more new *Delegates* have announced themselves to the new *Home Node*. As the majority of the increase in server performance when using PAMS comes from the collaborative cache’s ability to provide the prefixes for a relatively small number of popular content items, the loss of *Delegate* information for the most popular content will cause a spike in the bandwidth required by the server.

Churn also has a direct impact on the ability of overlay networks to successfully route messages to the correct destination. A high churn rate can have a significant effect on the stability of an overlay network.

As nodes leave the network, so too will the copies of prefixes for which they are *Delegates*. As there will be many *Delegates* for the prefix for the most popular content, the effect on server performance of losing a small number of these prefixes will be small.

Replication has been proposed in Section 6.1.2 as a means of reducing the effect of the loss of *Delegate* information when a *Home Node* departs from the collaborative cache.

Replication is used to ensure that, if a *Home Node* leaves the collaborative cache, the new *Home Node* will have enough knowledge of available *Delegates* to ensure prefixes can be supplied to clients and limit the effect of churn on the server.

In order to determine the effectiveness of replication, the reasons for a collaborative cache miss must be categorised. This is achieved in the simulation by using a "global overseer" that will determine why each collaborative cache miss has occurred.

Five categories of collaborative cache miss are proposed:

No Delegates: This category of collaborative cache miss will occur regardless of the accuracy of the directory information contained in *Home Nodes*, as there are no prefixes of the requested content in the collaborative cache.

No Bandwidth: Like **No Delegates** this category of collaborative cache miss will occur regardless of the accuracy of the directory information contained in *Home Nodes.* These misses occur when all *Delegates* are contained in the directory but none have available bandwidth to service the incoming request.

Lookup Failure: This category of collaborative cache miss occurs due to the failure of the underlying overlay network to locate the *Home Node* within the allowed period. Churn will increase the rate of lookup failures.

Some Unknown: Misses in this category result when all of the *Delegates* that are known by the *Home Node* have insufficient bandwidth to serve the requested prefix but there are other unknown *Delegates* that may have been able to supply the prefix.

All Unknown: This final category of collaborative cache miss occurs when *Delegates* exist but are all unknown to the *Home Node* for the requested content item.

Of the above five categories, the **Lookup Failure**, **Some Unknown** and **All Unknown** categories will be most affected by churn. In practice, simulation results have shown **Lookup Failure** misses to be very rare in the absence of churn. Furthermore, in the absence of churn, **Some Unknown** and **All Unknown** misses will only occur as a result of the small delay in *Home Nodes* becoming aware of the availability of new *Delegates*.

In this experiment, cache misses are recorded for varying churn rates. The benefit of replication is evaluated by comparing the overall collaborative cache hit rate when using replication to the hit rate that results without replication. To further understand the benefit of replication, collaborative cache misses are categorised using the five categories listed above. The parameters for this experiment are listed in Table 7.10

Parameter	**Value**
Mean time between churn event (Join or Leave) (seconds)	**5, 10 . . . 60**
Replication Policy	**No replication** **Replication every 60 seconds**
Number of Content Items	5000
Mean Stream Duration (seconds)	3600
Prefix Duration (seconds)	600
Cache Size (number of prefixes)	5
Mean Network Size (number of clients)	1200
Request Rate (requests/second)	0.25
Zipf Skew Factor	0.7
Local Cache Replacement	Least Frequently Accessed

Table 7.8: Experiment 8 Parameters: Effect of Churn

Discussion

The graph in Figure 7.19 shows the effect of churn on the collaborative cache hit rate both with and without replication. The y-axis shows the collaborative cache hit rate for each churn rate shown on the x-axis. The cache has the lowest hit rate when churn rate is highest, as would be expected. This is explained by a significant increase in the **Lookup Failure** misses, **All Unknown** and **Some Unknown** misses, as can be seen in Figure 7.20 (a).

By replicating a subset of the *Delegate* information stored by each home node to one additional node, as described in Section 6.1.2 in Chapter 5, some of the misses

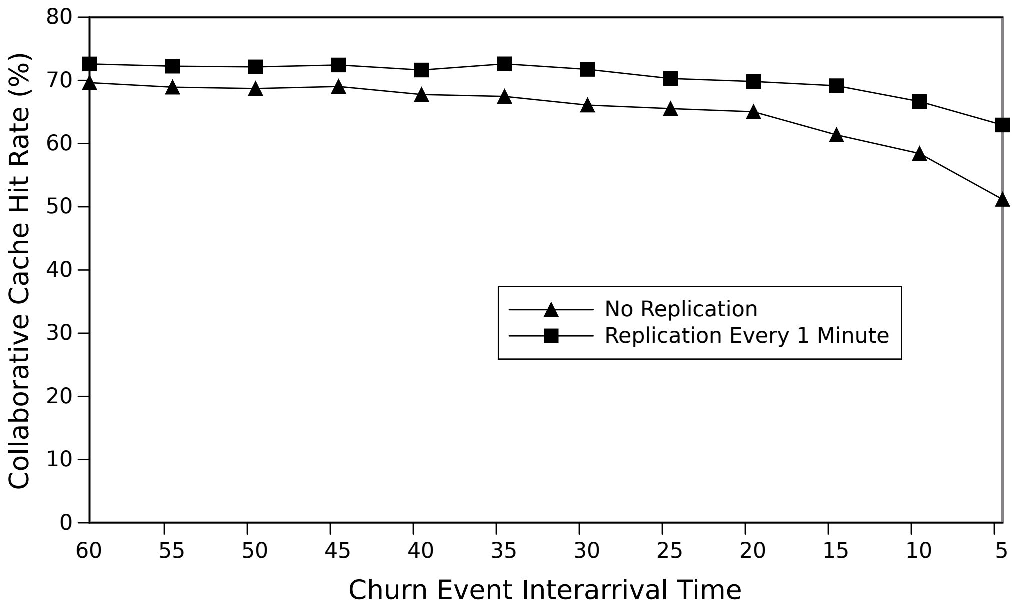


Figure 7.19: Experiment 7 – Effect of churn on collaborative cache performance

resulting from increased churn rates may be eliminated. This can be seen in Figure 7.20 (b), which compares the collaborative cache miss rates for systems without (a) and with (b) replication. The effect of replication is to reduce the overall collaborative miss rate by approximately 5.65% for a churn interval of 30 seconds. This will yield a corresponding improvement in server performance.

The most significant benefit of replication is in reducing the occurrence of **All Unknown** misses, that occur when a *Home Node* is unaware of any *Delegates* when unknown *Delegates* do in fact exist, as determined by the "global overseer" in the simulation. By replicating directory information to just one node, the occurrence of this category of misses is approximately halved.

Replication yields an increase in the **No Bandwidth** miss category. This is due to the *Home Node* being aware of all potential *Delegates*, but none having sufficient bandwidth to provide a prefix. The only way to avoid this miss is to allow clients to stream more than one concurrent prefix to requesting titles.

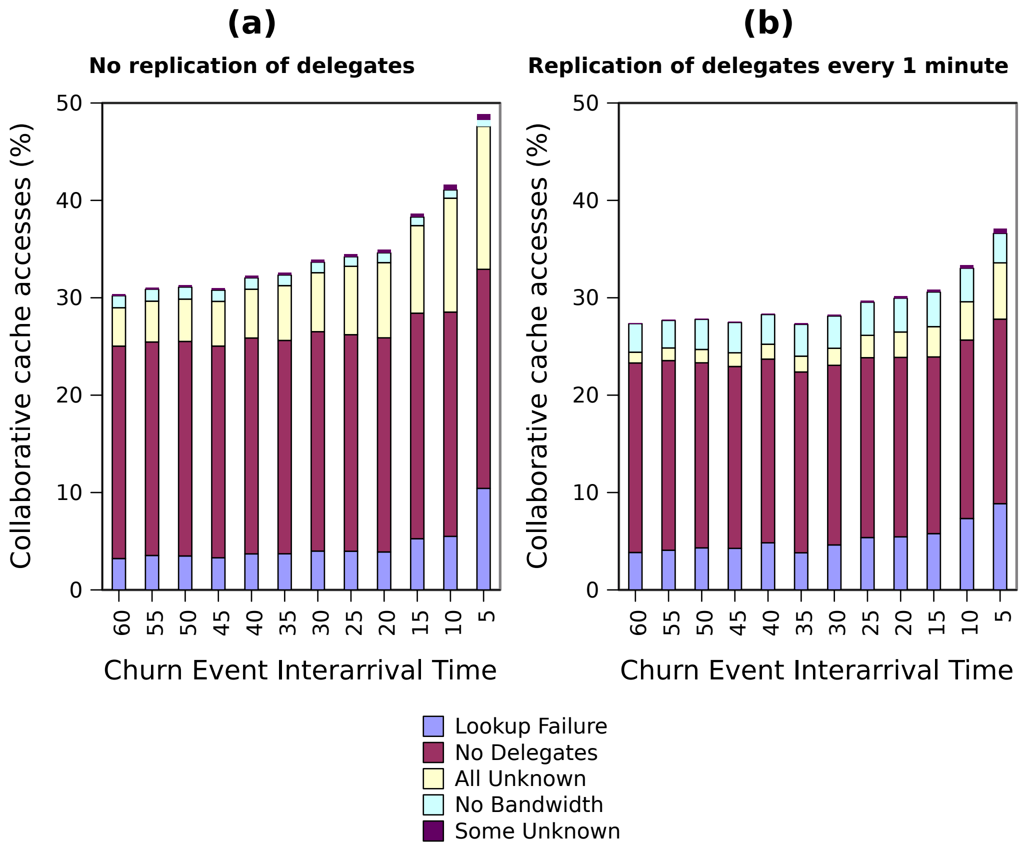


Figure 7.20: Experiment 8 – Effect of churn on collaborative cache performance

7.11 Summary

Experiment #1 compared the performance of Peer Assisted Multicast Streaming for prefix durations of 10 minutes and zero seconds, with a zero second prefix being equivalent to a simple unicast approach. By using PAMS with a ten minute prefix, the number of concurrent streams required by the server to satisfy all requests was reduced from 898.82 streams to 704.48 streams, representing a reduction of approximately 22%.

Experiment #1 also demonstrated that that most of the benefit of the PAMS approach resulted when serving the most popular content. For example, the number of concurrent streams required by the server to satisfy requests for five most popular content items was reduced from 62.78 streams to 16.82 streams, representing a reduction of 73.21%. This compares of a reduction of approximately 24.59% for the 96th to 100th ranked titles.

The benefit of using PAMS was shown to arise for two reasons. Firstly, by delivering prefixes of content items from the collaborative cache, the duration of streams provided by the server can be reduced. Secondly, by delaying the start of the server stream used to provide the remainder of a a content item, the server can group together multiple requests for the same content and satisfy these requests in a single multicast stream.

As the overall request rate increases, PAMS allows a server to batch together more requests into each individual multicast stream. As a result of this, the increase in the number of outgoing server streams will be less than the proportional increase in the request rate. This effect was shown in Experiment #2.

Furthermore, when the request rate for a particular content item is sufficiently high, the mean number of concurrent server streams required to serve this item reaches a maximum that is determined only by the duration of the content and the prefix duration. Any subsequent increases in the request rate for this item will have no effect on the number of concurrent streams supplied by the server.

By increasing the duration of prefixes provided by the collaborative cache, the batching window during which the server can group together requests for a content item is also increased. This results in an increase in the mean number of requests that can be served by each multicast stream, and a corresponding decrease in the overall number of concurrent streams required by the server. However, a trade off must be made as increases in the prefix duration will require additional client resources. This trade off has been examined in Experiment #3.

PAMS is most effective when there are a relatively small number of popular content items as this increases the request rate for those items and increases the probability that prefixes of these items can be served by the collaborative cache. As the distribution of requests for content items becomes more uniform, fewer requests for the popular items will arrive within the batching window. This will reduce the number of requests serviced by each multicast stream and increase the number of concurrent streams supplied by the server. The impact of changes in the relative popularity of content has been demonstrated in Experiment #4.

This increase in the uniformity of requests also impacts on the ability of the collaborative cache to serve prefixes, as it becomes less likely that a prefix for a requested item is contained within the collaborative cache. This results in a decrease of the prefixes served by the collaborative cache and a subsequent increase in the number of requests that must be satisfied completely by the server with little chance of other requests sharing the multicast stream.

The default cache size chosen in all of the experiments described in this chapter was sufficient to allow five prefixes to be stored by each client. This default cache size was chosen to be conservatively small. Experiment #5 however has shown that caches of this size are sufficient to yield a substantial improvement in performance over a simple unicast system. A further increase in the cache size will yield some benefit, however caching more than ten prefixes at each client yields little additional benefit but increases the overhead of maintaining the collaborative cache.

Experiment #6 examined the effect of varying the number of content items available to clients. Increasing the number of content items has the effect of diluting the number of requests for each item, reducing the effectiveness of PAMS by reducing the relative popularity of content. This reduction in relative popularity reduces the potential to group together multiple request for the same content item. Collaborative cache performance is also significantly impacted as its overall capacity is insufficient to cache the prefixes. Increasing the duration of the prefix and the size of local cache will counteract the effect of increasing the number of content items.

The final two experiments examined the ability of PAMS to adapt to the addition of new content and the removal of nodes. In the most extreme case, new content is added with the highest relative popularity, causing a “flash crowd”. Experiment #7 has demonstrated that PAMS will adapt quickly to flash crowds with a significantly lower peak in concurrent server streams than a unicast system. This peak was shown to be short lived.

Finally, Experiment #8 has shown that by replicating metadata within the collaborative cache, the effect of the churn rates examined is minimal.

7.12 Conclusions

Through the use of a detailed simulation framework PAMS has been shown to be effective in reducing the total number of concurrent streams required by a server delivering on-demand multimedia content. The reduction is centered around maximising the number of clients sharing a single multicast stream. The majority of the reduction is caused by clients sharing a small number of multicast streams for the most popular content item, where there is a maximum number of concurrent streams required to be provided by the server for an item. This maximum number of streams is determined by the prefix duration of the collaborative cache and the total duration of the content, and not by the request rate for the content as is the case in a unicast based system.

Chapter 8

Conclusion

This thesis has proposed a new architecture, called Peer Assisted Multicast Streaming, for delivering on-demand multimedia streaming services. The proposed architecture combines multicast and peer-to-peer technologies. The purpose of this new architecture is to both reduce the network load generated by a standard multimedia service and also to redistribute some of this reduced load towards the edges of the network. As on-demand multimedia streaming services, including television "catch-up" services such as the BBC iPlayer and on-demand movie services such as NetFlix, become more popular, the strain on server and network resources will increase. The work proposed here is intended to reduce this strain, particularly when supplying very popular content to large user communities.

Peer Assisted Multicast Streaming operates as follows. A client requesting a content item will first request the beginning or "prefix" of the content item from a collaborative prefix cache that the client is participating in. If it is successful, it will receive the prefix from the collaborative cache and will only need to request the remainder of the content from the server. If it is unsuccessful in obtaining the prefix from the collaborative cache it will request the entirety of the content from the server.

If a client can obtain the prefix from the collaborative cache the server can delay serving the remainder of the content. This will provide the server with an opportunity to service several delayed client requests with a single multicast stream. The server may begin the stream before all clients require the content and the receiving clients will buffer the stream until it is required.

The client will now receive the content in its entirety from either one or two streams. When a client receives a new prefix of a content item, it will add this to the collaborative cache making it available to other clients.

The use of the collaborative cache to provide a prefix results in the server being able to satisfy many requests for the same content item with a single multicast stream, without affecting the individual experience of each of the clients. This results in a reduction in the bandwidth required by the server to service all requests while still providing an on-demand service with the expected degree of client control. This reduction is greater for popular content as more requests will be received by the server during the delay period before the multicast stream must begin. Server resource utilisation is also reduced when serving less popular content items, even if only a single client is serviced by each stream, as the collaborative cache may be able to supply a portion (the prefix) of the content item. The collaborative cache also redistributes some of the network load away from the server towards the edges of the network, as clients provide each other with content that would otherwise have been provided by the server.

The collaborative cache is based on peer-to-peer technology, which brings some attractive properties. The collaborative cache is scalable and will grow and shrink in accordance with the usage of the system. The system is also responsive to changes in the popularity of content, as each client in the system that requests a content item will become a *Delegate* for that item.

The proposed implementation of Peer Assisted Multicast Streaming adheres to the guiding principles set out in Chapter 1. Firstly, a lightweight approach has been taken in the design and implementation of the collaborative cache. The approach taken to maintaining cache metadata sacrifices some cache performance to achieve a low overhead. For example, a more accurate collaborative cache replacement policy could have been implemented but this would have required additional network traffic, increasing the cache overhead. Secondly, the collaborative cache operates independently of the server. This avoids any additional load being placed on the server to facilitate the operation of the collaborative cache. (The only exception to this occurs when locally cached prefixes cannot be validated by the collaborative cache and must instead be validated by the server.) Thirdly, the proposed implementation uses existing network transport and control protocols. This means that a service based on Peer Assisted Multicast Streaming can be implemented without significantly changing the relationship or interaction between clients and servers and with only minor modifications to the operation of existing client and server streaming applications.

Peer Assisted Multicast Streaming has been extensively evaluated using a large-scale, detailed simulation. Simulation was seen as the most applicable method of evaluation as the benefits of multicast are only seen when the level of usage in a system is high. By building on existing network and peer-to-peer application frameworks for the OMNeT++ discrete event simulation system, it was possible to conduct a detailed performance evaluation that simulated activity down to the network packet level.

The performance evaluation has shown that, if the collaborative cache provides a ten-minute prefix of a content item that is sixty minutes long, a 22% reduction in the mean number of concurrent streams required to serve all requests is achieved, when compared to a unicast approach. This reduction was for a conservatively low request rate with an average of one request every four seconds. The reduction in the mean number of concurrent streams increases significantly with higher request rates. Also, as the request rate for the more popular content items increases, the number of concurrent streams serving that content item reaches a peak value. This means that the maximum amount of resources required to provide the most popular content is predictable and independent of the request rate for that content. The evaluations have also shown that, even while limiting the number of outgoing prefix streams served by each client to one stream and using a modest cache size of five prefixes, the collaborative cache would serve a prefix successfully for 70% of all requests. Furthermore it was shown that the collaborative cache responded quickly to the introduction of new content without significantly increasing the load on the server. Finally, the effect of clients leaving and joining the network, commonly referred to as “churn”, can be lessened by using a simple replication policy between the clients of the collaborative cache.

Each of the experiments described in Chapter 7 was of a large scale, with a minimum of 1,000 participating clients. Furthermore, most experiments evaluated the performance of PAMS over a range of parameters and each experiment was repeated numerous times to ensure the accuracy of the experiment. Each individual simulation ran for at least one simulated day and, depending on the parameters, could take between one hour or three days to complete. In order to run multiple parallel simulations, Grid resources [FKT01] were used. The experiments in Chapter 7 represent a total simulated time of approximately 730 days, which if run on a single processor would take approximately 60 days to simulate. Through the use of Grid resources, the time taken to run the experiments was significantly reduced.

While not discussed within the context of this thesis, there are still areas relating to multimedia content distribution that are worthy of consideration. One of the concerns is Digital Rights Management (DRM), and this is especially the case for PAMS as a peer-to-peer system is used to deliver some of the content. This can result in the content provider no longer having control over the distribution of the content unless a strong DRM policy is in place. Another concern not dealt with is malicious clients within the collaborative cache. For example, clients may intentionally provide corrupted files to the collaborative cache in order to breach the security of users of the system.

Peer Assisted Multicast Streaming was designed in such a way so that it could be easily integrated into existing on-demand multimedia services. For example, if a Content Distribution Network is used to distribute content over a wide geographic area, the servers at the edges of the CDN could be considered as multicast servers in a PAMS implementation. The clients accessing a single edge server would then participate in a PAMS collaborative cache.

A number of other architectures exist which combine peer-to-peer technology with multicast to provide an on-demand multimedia streaming service. These architectures, however, differ significantly from PAMS. In particular while the collaborative cache is PAMS in an entirely decentralized architecture, all of these existing architectures require a centralized server to locate buffered content and manage the activities of clients. Furthermore, the collaborative prefix caching approach that is proposed here caches content for long periods across multiple streaming sessions, thereby allowing more extensive use of peer resources. In contrast, in the existing architectures, clients only provide buffered content that they are currently receiving to other clients.

In order to implement Peer Assisted Multicast Streaming, the servers, the clients and the network connecting both will have to meet certain requirements. Servers would have to support multicast streaming and allow the start of a stream to be delayed until a time specified by a client. A client participating in a PAMS-based service would need to be able to request and receive two separate streams: one unicast stream from the collaborative cache and one multicast stream from the server. The client would also need to be able to merge these streams without any perceived break between the end of the prefix stream and the start of the server stream. There are additional client requirements resulting from participation in the collaborative cache. Each client will need sufficient storage capacity to store a small number of prefixes and sufficient outgoing bandwidth to supply a single prefix stream to another client. Each client must also implement the *Home Node* functionality for a number of content items. Finally, there will a small overhead resulting from each clients participation in the peer-to-peer overlay network and the management of the collaborative cache.

If the supporting network does not allow for multicast transmission then an alternative multicast approach, such as application level multicast amongst the participating clients, may be used. The main disadvantage of this is that each client will incur an increase in network, processor and memory utilisation.

8.1 Future Work

Although the lightweight approach for management of the collaborative cache helped reduce the network traffic overhead generated to maintain the collaborative cache, each client within the cache will only be aware of accesses made to its local copy of a prefix. As a result of this, the client will be unable to independently determine the relative importance of the prefixes in its cache. This prevents the client from implementing a cache replacement policy that maximises the global performance of the collaborative cache. This has been described in Section 5.3.5. Future work may investigate improved cache replacement policies that rely on clients sharing more information relating to cache accesses, thus enabling clients to make improved decisions. Such policies may come at a cost, however, as regularly updating each *Delegate* will result in more network traffic.

The popularity distribution of content is still an issue with Peer Assisted Multicast Streaming because when providing significant number of content items, only a small subset of these items will be popular. The remainder of the less popular content is usually referred to as the "long tail". For PAMS, serving the long tail only results in a reduction if the prefix can be served by the collaborative cache, as it is unlikely that more than one request will be received in the short period of time offered to batch together multiple requests.

For less popular content where a prefix is contained in the collaborative cache but it is unlikely that more than one request will be served by the multicast stream, a unicast stream should be used. This will reduce the cost involved in setting up and maintaining multicast routing. In order to provide the stream a unicast stream, the server will need to predict in advance how likely it is that another request will be received within a batching window. If the likelihood is low, the server will use a unicast stream to provide the content to the client. This can be achieved with standard RTSP control messages. Furthermore, the server may indicate to the client that the prefix need not be cached as it will be of little use to other clients, thereby improving the efficiency of the cache.

For all of evaluations performed the maximum prefix duration provided by the collaborative cache was predetermined. With variable prefix durations, longer prefixes could be provided for less popular content, increasing the likelihood of multicast sharing. This would also decrease the amount of content served by the main server even in the case where no multicast sharing is achieved. Determining the optimum duration for prefixes for each content item must take into account a number of factors, including the popularity of the content and how likely it is that by serving long, unpopular prefixes from the cache, there will be no free resources to serve more important, popular prefixes.

Within any peer-to-peer system it is likely that peers will exist who use the collaborative cache but do not contribute to it. This may be referred to as “freeloading”, and has been addressed elsewhere in the context of other peer-to-peer systems [Shr05] and may also be considered in the context of PAMS.

8.2 Concluding Remark

This thesis has shown that by using multicast and a small amount of client resources together, the network load created by an on-demand multicast streaming service can be both reduced and partially distributed around the network, without incurring a high overhead on the part of the server or the clients.

Bibliography

[AGP12] Mouna Allani, Benot Garbinato, and Peter Pietzuch. Chams: Churn-aware overlay construction for media streaming. *Peer-to-Peer Networking and Applications*, 5(4):412–427, 2012.

[And93] D.P. Anderson. Metascheduling for continuous media. *ACM Transactions on Computer Systems*, 11(3):226–252, August 1993.

[AT5] AT&T. AT&T latency statistics, 2015. Last accessed 19th September 2015.

[AWY96] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the 3rd International Conference on Multimedia Computing and Systems*, pages 253–258, Hiroshima, Japan, June 1996.

[Bha03] S. Bhattacharyya. An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational), July 2003.

[BHK07] I. Baumgart, B. Heep, and S. Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of the 10th IEEE Global Internet Symposium*, pages 79–84, Anchorage, Alaska, USA, May 2007.

[Bra99] K. Brandenburg. MP3 and AAC explained. In *Proceedings of the 17th AES International Conference on High Quality Audio Coding*, Signa, Italy, September 1999.

[CDG06] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443 (Draft Standard), March 2006. Updated by RFC 4884.

[CDK$^+$02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), October 2002. Updated by RFC 4604.

[CDK$^+$03] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 298–313, Sagamore, New York, USA, October 2003.

[CDKR02] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, October 2002.

[Coh03] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, California, USA, June 2003.

[DC90] S.E. Deering and D.R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, 1990.

[Dee89] S.E. Deering. Host extensions for IP multicasting. RFC 1112 (Standard), August 1989. Updated by RFC 2236.

[Den05] P.J. Denning. The locality principle. *Commun. ACM*, 48(7):19–24, July 2005.

[DFH99] S. Deering, W. Fenner, and B. Haberman. Multicast Listener Discovery (MLD) for IPv6. RFC 2710 (Proposed Standard), October 1999. Updated by RFCs 3590, 3810.

[dFR05] N.L.S. da Fonseca and H.K.S. Rubinsztejn. Dimensioning the capacity of true video-on-demand servers. *IEEE Transactions on Multimedia*, 7(5):932–941, October 2005.

[DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proceedings of the* 2^{nd} *ACM International Multimedia Conference*, pages 15–23, San Francisco, California, USA, October 1994.

[FAKM05] Z. Fei, M.H. Ammar, I. Kamel, and S. Mukherjee. An active buffer management technique for providing interactive functions in broadcast video-on-demand systems. *IEEE Transactions on Multimedia*, 7(5):942–950, October 2005.

[Fen97] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236 (Proposed Standard), November 1997. Obsoleted by RFC 3376.

[FGM$^+$99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[FIP95] FIPS. Secure hash standard (FIPS 180-1). Federal Information Processing Standard, April 1995.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, August 2001.

[FMAHK09] S. Farhad, Md. Mostofa Akbar, and Md. Humayun Kabir. Multicast video-on-demand service in an enterprise network with client-assisted patching. *Multimedia Tools and Applications*, 43(1):63–90, May 2009. 10.1007/s11042-008-0257-5.

[Gro10] The Nielsen Group. Nielsen three screen report, 2010. Last accessed 1st September 2011.

[HCS98] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the 6th ACM International Multimedia Conference*, pages 191–200, Bristol, UK, September 1998.

[HPL06a] K.H. Ho, W.F. Poon, and K.T. Lo. Enhanced peer-to-peer batching policy for video-on-demand system. In *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT '06)*, pages 148–151, Bangkok, Thailand, September 2006.

[HPL06b] K.M. Ho, W.F. Poon, and K.T. Lo. Peer-to-peer batching policy for video-on-demand system. In *Proceedings of the 1st International Conference on Communications and Networking in China (ChinaCom '06)*, pages 1–6, Beijing, China, October 2006.

[IRD02] S. Iyer, A. Rowstron, and P. Druschel. SQUIRREL: A decentralized, peer-to-peer web cache. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 213–222, Monterey, California, USA, July 2002.

[JKR02] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In *Proceedings of the 11th international conference on World Wide Web (WWW'02)*, WWW '02, pages 293–304, New York, NY, USA, May 2002. ACM.

[Koe99] R. Koenen. MPEG-4 multimedia for our time. *IEEE Spectrum*, 36(2):26–33, February 1999.

[LGJ$^+$06] C. H. Lee, Y. Q Gui, I. B. Jung, C. Y. Choi, and H. K. Choi. A peer to peer prefix patching scheme for VOD servers. In *Proceedings of the 3rd International Conference on Information Technology: New Generations*, pages 530–534, Las Vegas, Nevada, USA, April 2006.

[Lit61] J. D. C. Little. A proof of the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, 1961.

[LL97] W. Liao and V.O.K. Li. The split and merge protocol for interactive video-on-demand. *IEEE MultiMedia*, 4(4):51–62, October–December 1997.

[LRW03] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa network. In *Proceedings of the 3rd IEEE Workshop on Internet Applications*, pages 112–120, Sane Jose, California, USA, June 2003.

[OD09a] J.P. O'Neill and J. Dukes. On-demand multicast streaming using collaborative prefix caching. In *Proceedings of the 12th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS'09)*, pages 27–40, Venice, Italy, October 2009.

[OD09b] J.P. O'Neill and J. Dukes. Re-evaluating multicast streaming using large-scale network simulation. In *Proceedings of the 1st International Conference on Intensive Applications and Services*, pages 39–46, Valencia, Spain, April 2009.

[PD07] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach.* Morgan-Kaufman, 4th edition, 2007.

[PLF05] W.-F. Poon, K.-T. Lo, and J. Feng. Provision of continuous VCR functions in interactive broadcast VoD systems. *IEEE Transactions on Broadcasting*, 51(4):460–472, December 2005.

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[PWCS02] V.N. Padmanabhan, H.J. Wang, P.A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 177–186, Miami Beach, Florida, USA, May 2002. ACM.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralised object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*, pages 329–350, Heidelberg, Germany, November 2001.

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, pages 127–140, Boston, Massachusetts, USA, June 2004.

[RHKS01] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of the 3rd International COST264 Workshop on Networked Group Communication*, pages 14–29, London, UK, November 2001.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. Updated by RFCs 5506, 5761, 6051, 6222.

[SD00] Dinkar Sitaram and Asit Dan. *Multimedia Servers: Applications, Environments, and Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[Shr05] V. Shrivastava. Natural selection in peer-to-peer streaming: From the cathedral to the bazaar. In *Proceedings of the 15th International Workshop* *on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'05)*, pages 93–98, Skamania, Washington, USA, June 2005. ACM Press.

[SMK$^+$01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, pages 149–160, San Diego, California, USA, August 2001.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), April 1998.

[SRT99] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, volume 3, pages 1310–1319, March 1999.

[Tan96] A. Tanenbaum. *Computer Networks.* Prentice Hall, 3rd edition, 1996.

[Var01] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the 15th European Simulation Multiconference (ESM'2001)*, pages 319–324, Prague, Czech Republic, June 2001.

[VC04] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810 (Proposed Standard), June 2004. Updated by RFC 4604.

[Ver15] Verizon. Verizon latency statistics, 2015. Last accessed 19th September 2015.

[Wak08] J. Wakefield. BBC and ISPs clash over iPlayer, April 2008.

[WSBL03] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[XZZ$^+$04] Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, and Y.-Q. Zhang. Peer-to-peer based multimedia distribution service. *IEEE Transactions on Multimedia*, 6(2):343–355, April 2004.

[ZHS$^+$04] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

[Zip49] G. K. Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology.* Addison-Wesley, 1949.

[ZLLsPY05] X. Zhang, J. Liu, B. Li, and T. s. P. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, volume 3, pages 2102–2111, Miami, Florida, USA, March 2005.