1-1-2007

DistVid : an examination of a distributed video streaming system

Kevin Leung
Ryerson University

DistVid:
An Examination of a Distributed Video Streaming System

by

Kevin Leung, B.Sc Computer Science Ryerson University

A thesis
presented to Ryerson University

in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in the Program of
Computer Networks

Toronto, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

DistVid: An Examination of a Proposed Decentralized Video Streaming System
Master of Applied Science, 2007, Computer Networks, Ryerson University

Abstract

Despite the recent popularity of P2P file sharing strategies, on-demand video streaming have traditionally employed client-server type architectures or else multi-cast trees as the principle methods for multimedia delivery. However, as one may suspect, such systems have difficulties associated with the enormous cost when attempting the scale the system upwards to support more users. In light of the recent popularity of sites such as YouTube and Revver, a system that better utilizes existing infrastructure would be preferable. This thesis presents DistVid, a new protocol that was developed to combine the features of multi-cast trees and structured P2P overlays. The overall goal of this protocol is to provide a frame work for on-demand video streaming that is both resilient to failure and able to maximize available bandwidth. The work presented here will show that DistVid not only achieves this, but can also be employed to increase the efficiency of existing client-server systems or else replace them entirely.

Acknowledgments

I would like to express my gratitude towards Professor Lee for his guidance and insight which helped immensely in refining my thesis.
I would also like to thank Director Dr. Ma for his valuable comments and understanding.
Lastly, but certainly not least, to my wife and family for their patience, support and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Motivation & Aims

Infrastructure-based multi-media content delivery across networks have traditionally taken a client-server methodology. This generally involves delivering compressed audio and video from one, or possibly a few, server machines to potentially hundreds to thousands of clients. As a result, such servers, as a minimum. must have very large storage space, very high up stream and down stream bandwidth, and high reliability both in sense of the up time as a unit and the network infrastructure that provides access to the unit.

An immediate consequence of such a configuration is the high cost in setting up such "powerful" server machines and as a result the poor scalability of such an infrastructure. To support more clients, a substantial financial investment to establish another server would be required. [1] Another concern is the vulnerability to a single point of failure. Whether this occurs through physical

catastrophe, hacker attacks, or simply through being overwhelmed due to popularity, the disruption of a single point in the network, namely the server, can disrupt the service to a large portion of the network.

Peer-to-peer networking models have been proposed to solve many of the problems associated with the traditional client-server model. In the realm of multi-media content delivery, peer-to-peer communication in the form of multi-cast trees has been used to off load the high bandwidth demands on a single server. However, despite having been introduced over a decade ago and the ever increasing demand for multi-cast services, wide scale acceptance of network-layer multi-cast has been slow due to high cost, administrative and computational complexity and a lack of a coherent business model[14].

During roughly the same time period, the opposite was happening with peer-to-peer models. With its introduction in 1999, Napster had paved the way for technologies for leveraging the strengths of application level peer-to-peer networks for file sharing. To this day, the success of protocols such as Gnutella, eDonkey, and Bittorrent, at least from a file sharing perspective, cannot be contested[36].

In this thesis, an attempt is made to leverage the strengths of peer-to-peer application level multi-cast trees and peer-to-peer file sharing in a simple protocol named DistVid (Distributed Video). DistVid is a client driven protocol that seeks to minimize its dependence on servers by decentralizing peer coordination through the use of the distributed hash table (DHT) Chord. Much like peer-to-peer file transfer protocols, DistVid seeks to maximize the efficiency in which the network bandwidth is utilized by employing multiple peer, bittorrent style, transfers and minimizing its reliance on centralized servers. At the same time, by borrowing properties of application level multi-cast trees, network communication overhead is minimized.

1.2 Thesis Organization

This thesis will be organized as follows:

Chapter 1: The rest of this chapter will give further information on peer to peer networks, hashing, decentralized hash tables and finally briefly look at multiple description coding. PlanetSim 3.0 was used to model and simulate DistVid. Its simulation framework and programming API will be discussed briefly at the end of this chapter.

Chapter 2: Since DistVid implements the techniques of both Chord and Bittorrent, a more detailed look at each of these technologies will be provided.

Chapter 3: The DistVid protocol will be fleshed out, outlining the decision making process of the DistVid client nodes, how data will be retained, and the queuing strategies employed by parent nodes.

Chapter 4: Examines the effects of the different queuing strategies and DistVid's operation under a single parent multiple children model.

Chapter 5: Explores DistVid where multiple description coding is employed to allow DistVid nodes to take multiple parents.

Chapter 6: This thesis will be concluded with a look at possible future work.

1.3 P2P Systems - A Brief History and Overview

Under the notion of a pure peer-to-peer network, there are no client or server entities. Rather, nodes take on the functionality of ***both*** client and server in order to take advantage of the shared connectivity and resources amongst peers of the network. The concept of a peer-to-peer network, however, is nothing new. Its origins dates back to the mid 1960s with ARPANET, the world's first packet switching network and the predecessor of todays modern internet. With packet switching, it became possible to communicate with more than one machine at a time by dividing data into individual packets for transmission and subsequent reassembly at the receiving end. More importantly, it made possible the sharing of a single communication link and thus the economic feasibility of peer to peer connectivity. [2][6]

The first ARPANET link was established on January 14, 1969 between UCLA and the Stanford Research Institute (SRI). By December 5,1969 the remaining links with the University of California Santa Barbara, and the University of Utah's Graphics Department were established[3][4]. ARPANET, allowed researchers to use computers at various ARPA locations and assisted in collaboration efforts, making new software and research results widely available. By accomplishing this without any form of centralized control and coordination, the four ARPANET IMP (Interface Message Processors) nodes created the first four node peer-to-peer network.

While there are some well established peer-to-peer structures in the current day internet, such as the communication amongst domain name service servers and Usenet news servers, the internet has

generally progressed favouring client-server based models. The main advantage of peer to peer networks over traditional client-server models is the ability to take advantage of the cumulative resources of a large group of nodes on the peer-to-peer network, such as, bandwidth, storage space and computing power.

However, there is a price that must be paid for this increased functionality. Namely, the difficulty in coordinating the various individual peers of the network to work as a whole. This has led to a classification of peer to peer networks based on the strategies used to organize the member peer nodes. Centralized peer-to-peer networks, such as Napster, utilizes a central server to construct pointers and to resolve address lookups by peers. Decentralized peer-to-peer networks, as its name implies, has no centralized server and is further divided into two categories. Structured and unstructured[5].

An unstructured peer-to-peer networks such as Gnutella, have nodes maintaining open connections to at least one other node on the network. Queries and group maintenance are achieved by flooding messages to all known nodes. Because of this, there have been concerns as to the scalability of such protocols[8][9].

Structured peer-to-peer such as Chord [10], CAN [11], and Pastry [12] work by mapping object keys to the responsible overlay nodes and providing functions to locate and contact those nodes. Structured peer-to-peer overlays are called such because they generally organize participating nodes into positions along geometric structures such as circles, hyper-cubes, and toruses. This is achieved by assigning nodes an ID from a known identifier space that correlate directly to a position on the geometric structure. Data objects are identified by keys selected from the same identifier space as the node identifiers. The nodes whose ID most closely matches a particular object's key becomes responsible for that object. Because data sets are mapped to nodes whose position are predictable, locating a node and its respective contents can be achieved, if not easily, then at the very least

predictably. [13]

1.4 Hash Tables

Hash tables are used to construct abstract data types such as associative arrays, look-up tables, maps, and dictionaries, where values are related to a specific key. At the core of a hash table is the hash function h which takes a key x and maps it to a hashed value $h(x)$. Ideally, $h(x)$ is an a value that can be mapped directly to an unique position in an array space[15][21].

The goal of a hash table is to provide key based lookups at near O(1) constant time. However, this is only true in cases where $h(x) \neq h(y)$ when $x \neq y$ for some hash function h. When two or more keys hash to the same value a collision is said to have occurred. A container to store multiple keys mapped to a particular hash value is termed a hash bucket. The method to uniquely identify each value contained within a bucket is the collision resolution scheme.

A common collision resolution scheme is one that employs the use of lists in a process named chaining. Upon recovering a hashed value $h(x)$ an algorithm would search linearly down the list, stored within the bucket, until the data associated with key x is found. A poor hash function that map a large portion of values to the same hash bucket can result in a O(n) linear look up time that would scale poorly for large n.

1.5 Distributed Hash Tables

A distributed hash table (DHT) is an extension of the basic hash table with the same principal goal of providing a lookup service. That is, to be able to quickly locate, insert, and delete large amounts of data quickly. Like regular hash tables, data is once again associated with keys which in turn are mapped to a hashed value with an hashing function or algorithm such as SHA-1.

The difference however is that now, the concept of a bucket, has been replaced by a physical nodes that are dispersed throughout a network. Nodes are assigned IDs from the same identifier space as the values generated by the hashing algorithm. The node whose ID is the closest to the hashed key becomes responsible for storing the the key and data pointer pairs. Collision resolution takes place within the responsible node and the nodes themselves may implement local hashing tables to retrieve the information associated with the sought after keys[10].

Older peer-to-peer models such as Gnutella and Freenet, avoided having a single point of failure because of their decentralized nature. However, compared to traditional client-server approaches they had significant disadvantages. Gnutella, with its flooding based query model, paid a heavy price in network traffic efficiency. Freenet uses a key based routing protocol instead of flooding, which significantly reduces network traffic. However, because the routing algorithm is based on a heuristic seeking cached copies instead of a clear separation of responsibility amongst nodes, there are no guarantees that the available data will be correctly located[16][17][10].

Like Freenet, DHTs uses key based routing as well. However, the difference is that DHTs

routes messages based on a geometric structure defined by the virtual positions of participating nodes. These nodes have a clearly defined region of responsibility in the ID space. Thus, routing queries to the correct destination is much more direct and efficient potentially providing guarantees approaching that of a centralized protocol such as Napster. At the same time, these protocols remains entirely decentralized giving it the robustness and fault tolerance not available to a server based models. Figure 1.1 illustrates a generic DHT where geographically diverse nodes connected via the internet are responsible for a subset of key-value pairs.

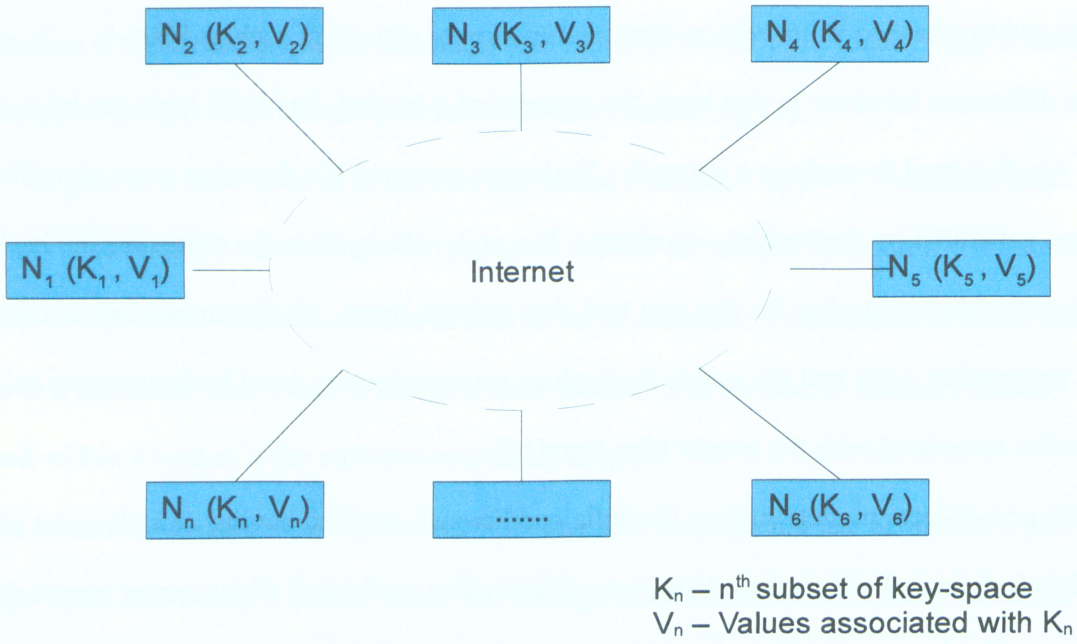


Figure 1.1 Decentralized Hash Table

A DHT retrieval system is not without its share of difficulties however. Lookup latency can scale from an unsustainable O(n), where a query may be required to visit all participating nodes before locating its destination, to a much more efficient $O(\log_2 n)$. Since each node participating in the DHT becomes an overlay router, participating in both data transfer as well as routing lookup requests, higher resource requirements, such as, memory and processor speed, must be accounted for. Geography is

also a concern. In the case of the Chord DHT, a node's position on the overlay network can potentially have no correlation to its actual physical geographic location. Two nodes that may be virtually close together, by their overlay addresses and thus communicating frequently, can easily be continents apart physically. Finally, churn, where participating nodes join and leave the DHT rapidly, can result in the partitioning of the system. This in turn will slow the recovery of the overlay and ultimately interfere with the system's ability to deliver messages and queries correctly.

1.6 Consistent Hashing

The concept of consistent hashing applies to both traditional hash tables and distributed hash tables schemes whereby the addition or removal of "slots" (i.e. table space) does not significantly alter the mapping of keys to the rest of the table. In other words, a consistent hashing function is one that experiences minimal changes as the range of the function changes[16]. This compares favorably to traditional hashing strategies where a change in the number of array slots can result in nearly all keys being remapped[28][10].

Consistent hashing is an especially important concept in decentralized structures such as DHTs where nodes may enter and leave the network unpredictably and thus changing the number of available buckets/slots to the DHT. Having to reassign/remap all keys on the network to different nodes each time a node leaves or enters can potentially render a DHT unusable as a large amount of time and network resources are consumed in updating key positions.

Ideally, with consistent hashing, each node is responsible for at most $(1 + \varepsilon)K/N$ (for an arbitrarily small value ε) keys for a network of N nodes and K keys. As the a node enters or leaves the

network the number of keys that change hands should be bounded by O(K/N)[10][31]. Thus only a small subset of keys are affected by the arrival or departure of a node minimizing both the amount of peer communication required to stabilize the overlay network and the number of nodes affected.

1.7 Multiple Description Coding

One the main reasons file sharing is able to take advantage of the strengths of peer-top-peer networks lies with the possibility of fragmenting the file into smaller discrete units. Whether it be a bittorrent "piece" or an eDonkey "chunk" the concept remains the same. By dividing a file into smaller segments, a peer-to-peer protocol is able to take advantage of network path diversity and the upload bandwidths of multiple nodes. A node seeking a particular file seeks multiple nodes that have that file available and then downloads different pieces from each of those different nodes.

While this model can be allied to one seeking a whole video, say a previously encoded movie, this model fails for real time or on-demand multimedia streaming. The principle reason for this is that video streaming is sequential and time dependent. Video frames that are in the future of the current viewing position serves at best as buffered video and video frames that arrive late might as well have not arrived at all.

Multiple description coding (MDC) is a well established coding technique whereby a video stream is fragmented into multiple independent sub streams. For decoding, any of the sub streams can be used. However, the more sub streams recovered for a particular frame or groups of frame the higher the quality of the decoded image. Traditional progressive style video transmission works well when packets are sent and received in order without packet loss. However, in the presence of packet loss,

multiple description coding has been shown to get a use able image to the end user much quicker. [20] An example of a simple two stream MDC is to split a source video into two streams having the first stream encode only the odd horizontal rows of a video frame and the second stream the even rows. When only a single stream is available, an approximation of the original image can be generated by interleaving the two rows adjacent to the missing rows.

The advantages of MDC are multi-fold. First, by encoding lower bit rate sub-streams, a lower burden is placed on the streaming nodes upload bandwidth. Secondly, nodes seeking the video stream can take advantage of the upload bandwidth of multiple nodes and take advantage of path diversity by seeking different sub-streams from different nodes. Indeed, the work of Xu et.al. has shown that for non overloaded networks, there is a significant improvement in system performance by using more lower bandwidth sub-streams than fewer sub-streams of higher bandwidth. [1] It was concluded that this improvement comes directly from the increased path diversity as a result of having more sub-streams.

MDC's advantages are not without limits however. As a video is broken down into more and more sub-streams, both encoding complexity as well the underlying network transport overhead increases. Eventually, multiple description coding may no longer be worthwhile as compared to traditional encoding methods[38].

Figure 1.2 illustrates one of the main advantages of MDC. In a heterogeneous network environment, MDC allows for the provision of differentiated service to best match a client's available bandwidth. With higher bandwidth clients, more sub-streams are provided to increased video quality while lower bandwidth clients will still receive service albeit at a lower quality.

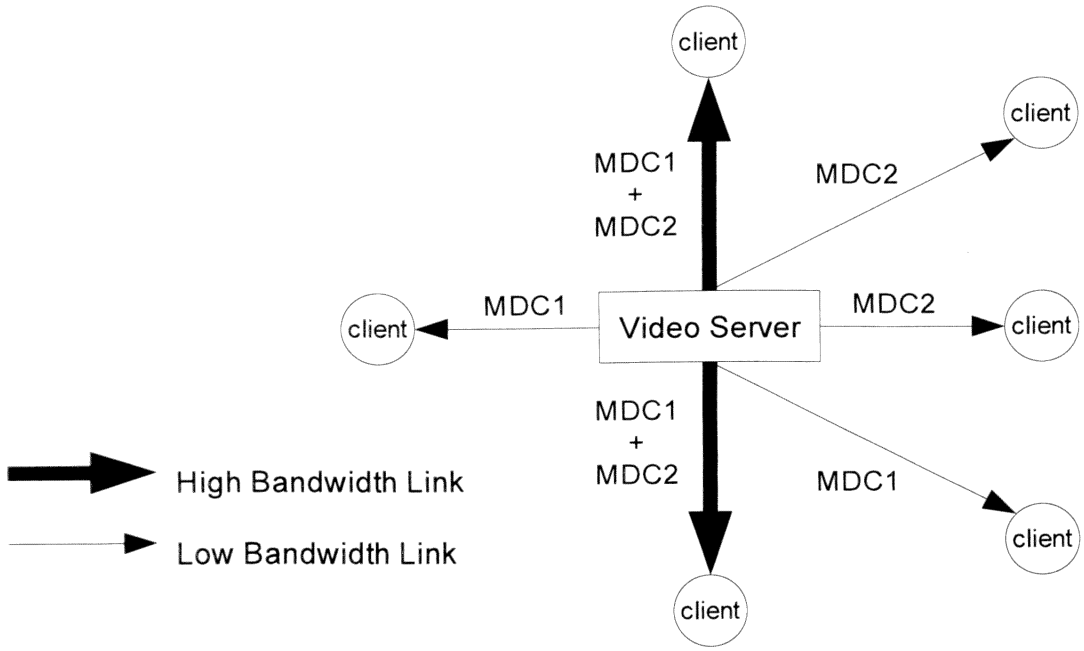


Figure 1.2 Multiple Description Coding Distribution

1.8 PlanetSim 3.0

The results presented in this thesis originate from an implementation of DistVid on the PlanetSim 3.0 simulator. PlanetSim is a Java based discrete event simulator packaged with implementations of Chord and Symphony. The strengths of PlanetSim lies in its Common API (CAPI) which allow for a decoupling of the peer-to-peer application from the underlying DHT overlay algorithm. This is accomplished by having the CAPI establish three tiers, 0, 1, and 2 corresponding to the network, overlay, and application layers respectively. By doing so, studying a peer-to-peer application on different DHT overlay algorithms become a simple matter provided the overlay has been implemented. Communication between the three tiers is accomplished through up-calls and down-calls provided by the CAPI[23].

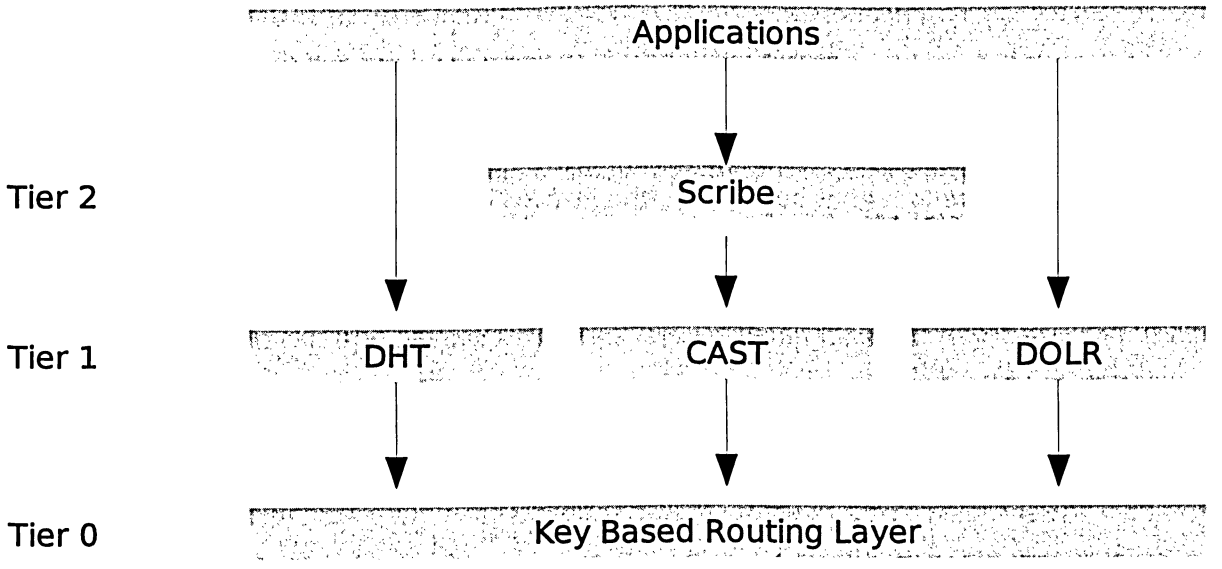


Figure 1.3 PlanetSim API

The authors of PlanetSim have indicated their intentions to implement TCP/IP wrappers for PlanetSim's CAPI. Once complete, it will be possible to take the same code used within the simulator, and with litter to no modification, have it executed on real TCP/IP networks, such as PlanetLab. Verification of simulated results would then be a simple matter of comparing results. DistVid, for the moment, exists on the application level. However, with the introduction of Vivaldi network coordinates to the Chord DHT, modifications of the Tier 1 API are planned to forward network coordinate values to the application level. This will allow applications such as DistVid to locate "closer" lower-latency peers when seeking to form peer relationships.

PlanetSim provides an implementation of Chord and a well defined Java API making it a valuable platform for providing insight as to how DistVid will operate in the real world at the application level. However, with that said, it should be noted that PlanetSim does ***not*** model the actual network traffic at the packet level. Thus, it does not simulate the effects of real world link latencies, packet loss, network congestion, etc[23].

Chapter 2 Related Work

The distributed video protocol presented in this thesis attempts to combine the strengths of multi-cast tree style forwarding, bittorrent style file transfer, and the decentralized tracking abilities of Chord. For this reason, in this chapter, an overview of the functionality of Chord and bittorrent is provided. However, it should be noted, both are complex protocols in themselves and a full analysis of each is beyond the scope of this thesis.

2.1 Chord

Chord is a decentralized peer-to-peer lookup protocol which forms the foundation on which DistVid will be tested. It is designed to be scalable in an environment where there are frequent node arrivals and departures. Compared to DHTs like Pastry, by caching minimally, Chord favours lookup accuracy at the expense of latency[10]. Chord, as a structured P2P network, maps its member nodes onto a unit circle and provide four distinct advantages: 1. Simplicity in structure and thus

implementation; 2. O($\log_2$ n) lookup time; 3. the ability to handle random joins and departures of many nodes simultaneously, and; 4. has no reliance on a centralized server.

Some potential drawbacks with Chord include the assumption that there will be no malicious nodes on the network. Examining the Chord algorithm, detailed in the following section, it is readily apparent that a fake node that forwards lookups improperly can seriously disrupt the ability of other nodes to perform lookups[10]. There is also an underlying assumption that a system is in place for any joining node to find a node already on the Chord network. Without such a system, join operations are not possible.

2.1.1 Chord Algorithm

To fulfill its lookup functionality, Chord needs to be able to forward lookups to nodes correctly. To accomplish this, Chord nodes maintains a finger table of a subset of the nodes known to be on the network. The size of a finger table is at most $\log_2$ n neighboring nodes with Chord purposely restricting its knowledge of the rest of the network. At the expense of lookup latency, this serves to minimize the necessary resources at each node and the amount of network traffic necessary for book keeping purposes. This is especially useful in configurations where participating nodes can be quite restricted in both computing power and available memory, and when the reliability of nodes are questionable. During a lookup, a node checks its finger table to determine the best node to send queries to and depending upon it to further forward the query as necessary. This target node is either the destination node itself or its closest known predecessor.

Chord is dependent its hash function to balance load across the ring such that each node responsible for approximately the same number of keys. In Chord's case, this is generally SHA-1. SHA-1 is an algorithm that until 2005 has been thought to be unbreakable from a practicality standpoint and thus considered sufficiently collision free[29]. As it stands, the works of Wang and Yao

has shown that at least 2^{69} operations are required for collisions to occur[30]. SHA-1 is a good choice for Chord as a hashing algorithm because of its ability uniformly distribute values within the address space. This is important in preventing clustering and hot-spots in the Chord algorithm.

Chord uses m bit identifiers for the identification of its participating nodes and for determining the location of keys. When using SHA-1 as the base hashing algorithm, the size of m would correspondingly be 160 bits. A node's ID is generated by hashing the node's IP address along with it's port number. This ID then determines the node's virtual location on the Chord ring with higher valued Ids being placed clockwise to lower IDs. Note that the node's virtual location on the overlay has no correlation to it actual physical geographical location.

A key k can be any value and is usually assigned by the P2P application running on top of Chord. It is hashed into Chord ring's address space with the same base hashing algorithm. Key assignment is managed in a clockwise fashion. The key space is always larger than the number of physical nodes with keys being assigned to the first node whose identifier is ***equal to or follows*** the hashed value of k. This node is referred to as the ***successor*** of k.

Such a simple mapping of the key space to the node ID space and the tracking of the successor is all that is required for Chord to fulfill the requirements of consistent hashing discussed in section1.6. In the ideal situation, a leaving node would be responsible for roughly K/N keys. When this node decides to exit the network cleanly, it need only transfer K/N keys to its immediate successor for the DHT to continue operating normally[10].

2.1.2 Finger Table

Locating the node responsible for a certain key k becomes a matter of locating the k's successor node. This is easily accomplished simply by having each node on the Chord ring track its successor on the ring. This is the next node clockwise on the ring. A lookup message by a particular node can then be passed sequentially around the ring until the node responsible, k's successor, receives the message.

While this is sufficient to resolve a lookup, it is far from efficient since, for a Chord network of N nodes, a message may need to traverse all N nodes before reaching the responsible node. To accelerate this process, Chord nodes a finger table. This finger table has at most m entries with m, as mentioned earlier, being the number of bits used for the node IDs. A finger table entry includes the target node's ID and its IP address with the first entry being the current node's immediate successor on the chord ring. The work of Gupta et. al. discusses the feasibility of a maintaining a *complete* routing table to all nodes[25]. However, the restricted table size was intentional on the part of the Chord developers as a balance between routing efficiency and excessive network overhead.

The finger table is structured such that the i^{th} entry in the table for a particular node n identifies the successor node whose ID either equals or follows the result of the computation $(n + 2^{i-1})$ modulo 2^m where $1 \leq i \leq m$. Example finger tables for nodes with IDs 2 and 11 are presented in Figure 2.1, Table 2.1 and Table 2.2 for a chord ring with 6 nodes and $m=4$.

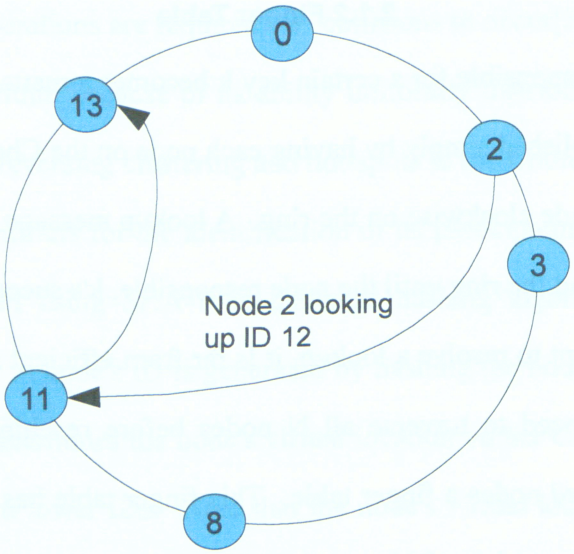


Figure 2.1 Chord Forwarding

Node 2				Node 11			
Entry	Start	Interval	Successor	Entry	Start	Interval	Successor
0	2+1 = 3	[3, 4)	3	0	11+1 = 12	[12, 13)	13
1	2+2 = 4	[4, 8)	8	1	11+2 = 13	[13, 15)	13
2	2+4 = 8	[8, 10)	8	2	11+4 = 15	[15, 3)	0
3	2+8 = 10	[10, 2)	11	3	11+8 = 3	[3, 11)	3

Table 2.1 Chord Finger Tables

Node ID	Key Responsibility Range	Number of Keys
0	(13, 0]	3
2	(0, 2]	2
3	(2, 3]	1
8	(3, 8]	5
11	(8, 11]	3
13	(11, 13]	2

Table 2.2 Key Responsibilities

In this case, node 2 seeking the node responsible for the ID 12, resolves from its finger table that node 11 is the next hop. It sends its lookup query to node 11 which in turn refers to its own finger

table to see that ID 12 is the responsibility of node 13. The query is thus forwarded correctly to node 13.

2.1.3 Load Balancing

From Figure 2.1, one can easily see that Chord can suffer from a problem common to many DHTs. Namely, there can be a large disparity between the overlay topology as compared to actual physical geographic topology. It is entirely conceivable that two "close" nodes on the overlay are in actuality continents apart. Recent implementations of Chord use the Vivaldi[27] decentralized network coordinate algorithm to more efficiently window their messages to take into account link latencies which have been shown to correlate well with geographic distances[26][27].

Another potential problem is how evenly nodes are distributed throughout the ring determine how many keys a node becomes responsible for. Close grouping of nodes on the ring id space can result in nodes assuming higher burdens than its peers resulting in potential hot spots. Even with only a 4 bit address space, this effect can be seen in Table 2.2 on node 8 which is responsible for an disproportional number of keys. Optimum load balancing is achieved when nodes are evenly distributed throughout the ring. In the ideal case, for a Chord ring with N nodes, if one were to evenly divide the Chord identifier space into N slots, there would ideally be one node in each slot. In such a configuration, keys would be distributed in such a way that, the number of keys per nodes is roughly constant throughout the Chord network[10]. It has been suggested for future work, that associating keys to virtual nodes and then mapping multiple virtual nodes to a single physical node can aid in load balancing.

2.1.4 Node Joining/Leaving/Churn

In a peer to peer network, one must expect nodes to arrive and leave unpredictably. The term ***churn*** is used to refer to the near constant arrival and departure of nodes from a distributed network. In light of the fact that churn can cause routing inaccuracies that can ultimately bring the entire overlay network down, it is critical that a DHT can manage and recover from churn[24].

The work of Rhea et al. looks at two methods for recovering from churn, reactive and periodic. Reactive recovery involve strategies whereby DHT nodes attempt to find a suitable replacement neighbor as soon as it notices that its existing neighbor has failed. Chord utilizes what is referred to as a periodic recovery technique whereby every node will periodically run a stabilization algorithm. It is through this stabilization algorithm that new nodes get integrated into the Chord network and links to failed or exited nodes purged.

When a given node n enters the Chord network with an ID between its predecessor n_p and its successor n_s, n would adopt n_s as its successor and notify n_s that it is now its predecessor. When it is time for n_p to invoke its stabilization routine once again, it will ask its current successor n_s for it predecessor which is n. n_p now acquires n as its successor who in turn is notified that n_p is its predecessor[10]. The work of Stoica et. al. proceeds to prove that using this algorithm problems that may arise from the concurrent joining of multiple nodes, such as when two nodes enter between n_p and n_s before any of the nodes have had a chance to stabilize, are temporary[10].

Node failures operate on a similar manner. Ideally, when a node leaves the network it will ensure all the keys that it is responsible for is transferred to its successor and its departure communicated to its successor and predecessor such that their fingers can be updated correctly. In the realistic operation of a distributed system one must expect nodes to depart unexpectedly and not cleanly. A Chord node's periodic invocation of its stabilize algorithm then becomes responsible for repairing failed fingers of the remaining participating nodes.

Reactive recovery, although found to have lower latency and bandwidth requirements, was actually determined to be less efficient than periodic recovery under "reasonable", Kazaa like, churn rates in conjunction with being vulnerable to positive feedback cycles[33]. Positive feedback cycles is a situation where neighbor nodes are falsely viewed to have failed or have left the network due to communication time outs resulting from network congestion. These nodes, believing their respective neighbors to have failed, then proceed to update their routing tables with incorrect information and further communicate that information to their new perceived successors and predecessors. This communication not only conveys erroneous information but also detrimentally adds to the burden of an already overly congested network link.

Periodic recovery is thus preferred for realistic churn rates. However, periodic recovery is susceptible to having a large percentage of long distance links become outdated and erroneous as the overlay topology changes. As can be seen from a Chord node's finger table, much more is known about nearby neighbors than far away neighbors[10]. Thus, it has been proposed that, for moderate to high churn rates, periodic recovery be used but that stabilization routines are run more frequently for long distance fingers than for closer fingers[24][33].

2.2 Bittorrent

Bittorrent is a popular peer-to-peer application designed for the replication of large amount of fixed data. It differs from traditional peer-to-peer file sharing protocols in that instead of requiring/encouraging users to maximize the variety of content being shared, bittorrent establishes a new peer to peer file sharing session, called a torrent, for every distinct item that is shared. In essence, for every item that is desired or shared, a downloading/uploading node becomes a member of another peer-to-peer transfer overlay[34].

A file that is to be distributed under this protocol is first divided into multiple pieces with each piece identified by its SHA-1 hash.

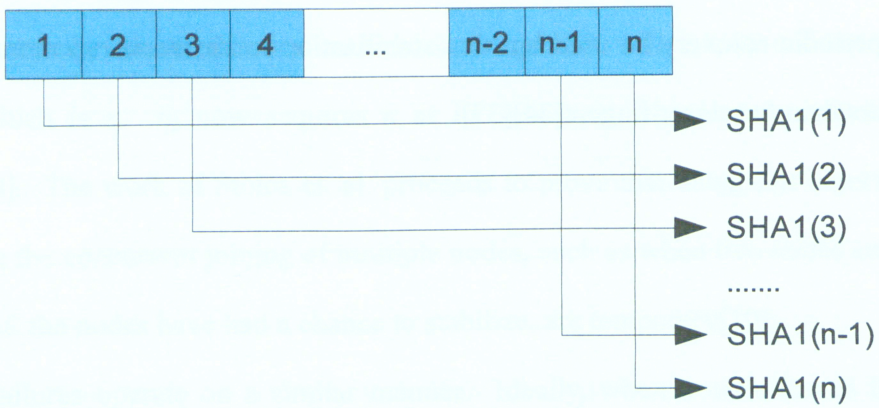


Figure 2.2 Bittorrent Hashing

Through a torrent file, meta-data consisting of the the number of pieces, hashes of all pieces, the hash to piece mappings, and, the responsible tracker, is distributed to all interested peers. The tracker is the centralized server that is responsible for collecting statistics and keeping track of all nodes participating in the distribution of a particular file. It does not directly participate in the distribution of

the file.

The set of all nodes participating in the distribution of a particular file is referred to as a swarm. When a node wishes to participate in a swarm, it first contacts the tracker identified in the meta-file. The tracker then provides the joining node with the IPs of a subset of the existing swarm chosen at random. This is usually 50 nodes. The new node and the 50 peers it is aware of will form symmetric relationships, uploading *and* downloading from each other, to distribute all pieces of the desired file.

The ultimate goal of the bittorrent protocol is to efficiently achieve a state where all nodes in the swarm have all pieces of the file being shared[34]. To achieve this, bittorrent implements a "rarest first" algorithm. Nodes have no mechanism to determine the global distribution of pieces throughout the swarm. Rather, to approximate it, a node will communicate with the subset of the swarm that make up its neighbors to determine the local piece availability. From the local information, the node will then rank pieces that it does not yet have based on the number of copies available. It then tries to maximize the number of *distributed copies o*f the rarest piece by downloading from the set of rarest pieces. Piece selection amongst the rarest set is performed randomly.

Figure 2.3 illustrates the node relationships amongst Bittorrent nodes for the simple case where there is one source for every required piece. The greyed out pieces represent available pieces with the lighter grey squares representing the piece(s) in the rarest set between that particular node and the new node. The total download bandwidth that the receiving peer experiences is then the aggregate bandwidth of all incoming transfers from peers A through F.

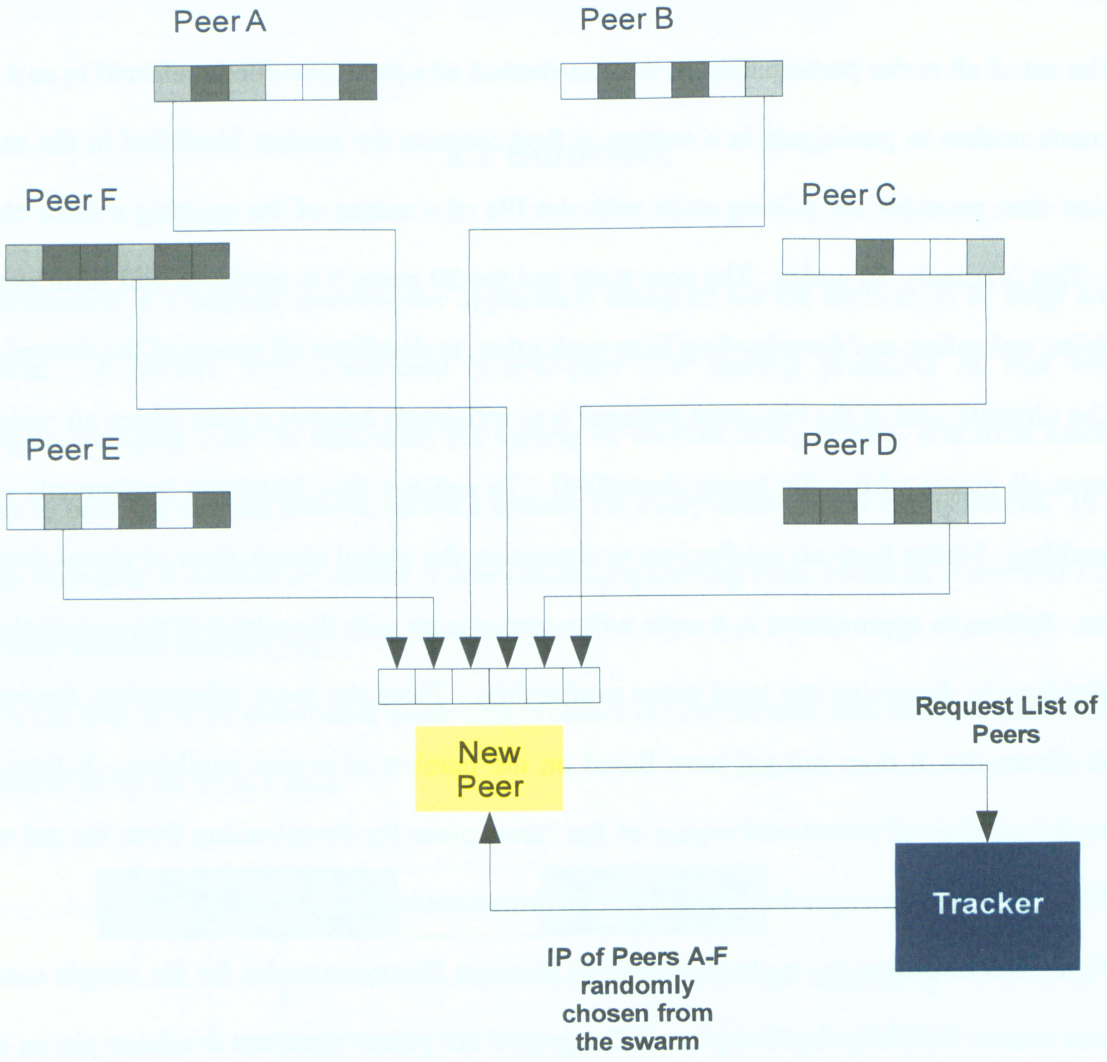


Figure 2.3 Bittorrent Swarm Downloading

Chapter 3 DistVid Protocol

In a file share system, the emphasis is on availability of static data both in the sense of being able to find where the data is stored and the ability of the system to recover the data especially in the face of disruptions in the network. While traditionally organized and maintained via centralized servers, a DHT , due to its decentralized nature, can provide fault tolerance, minimize traffic hot-spots and decentralize the tracking of node and data. In recent file sharing protocols, such as bittorrent, we also see a much better network bandwidth utilization than traditional one to one peer to peer transfers. Instead of a single 40kB/s transfer, a bittorent client can obtain different parts of a file from different peers to achieve a much higher aggregate bandwidth.

Multimedia streaming, in contrast, has a very different emphasis. The goal of a media streaming strategy is to minimize jitter, latency, and maximize visual quality. Indeed, it has been shown that maximizing file availability through techniques such as replication, while an effective strategy for file sharing, is often not the best strategy when constraints such as bandwidth comes into play[35]. Thus, popular methodologies for media streaming are usually achieved through direct server-client relationships, or else, by leverage the strengths of P2P transfers in the form of multi-cast trees.

The benefits of a tree based system include, minimal protocol overhead at individual nodes, simplicity of implementation, and a predictable latency for video stream arrival.

As can be expected, multi-cast trees are not without their own limitations. VCR style functionality is difficult. To recover differing parts of the video stream entail a node moving to a different part of the multi-cast tree. If one were to, for the time being, put aside the inherent difficulties of moving a node within a tree, without some form or centralized tracking, the act of locating the appropriate node to move to would itself be difficult. Centralized tracking, however, creates the potential for a single point of failure for the entire system. Multi-cast trees are also greatly disrupted by the effects of churn and, similar to structured overlays, can have poor correlation between tree positions and actual physical geographic locations.

DistVid is a simple protocol that attempts to combine the strengths of peer-to-peer file sharing and application layer multi-cast trees for the end goal of video streaming. The protocol is client driven, and data centric where most of the control is delegated to the individual clients, instead of servers, in their search for relevant data. It is the responsibility of the client to determine the availability of the desired data *and* to establish the connections to obtain the data. There will be little to no centralized tracking and management. The nodes participating in the video stream determine for themselves where and how to join the stream.

To achieve this, DistVid treats video as a hybrid between a multi-cast stream and a shared file under bittorent. Because DistVid is decentralized in nature, it will be reliant on a DHT for information tracking. While, technically, any DHT will do, Chord was chosen to be the DHT substrate on which DistVid would be tested primarily because of its resilience to churn. This is an important consideration for our purposes since in a video streaming environment as it is conceivable that large number of nodes may enter or exit simultaneously.

The function of Chord, in DistVid, is to map a key onto a node. A node can be anything from a

PC, server, to a PVR box on top of a TV. Each file on the Chord network has a unique key associated with it that is generated though consistent hashing techniques. This is the SHA-1 hash of a label that identifies the file or perhaps of the file itself. The Chord algorithm then makes the node, whose overlay address succeeds the key, responsible for that key. Thus it is possible to map keys to specific values. In our case, Chord will be used to map the physical addresses of nodes to the parts of the video that we seek.

DistVid is targeted for a heterogeneous networking environment, common in today's internet, where there are many participating nodes but most nodes while having relatively high download bandwidth, may have limited upload capacity (e.g. 600kbps). Examples of such connections include asymmetric cable and DSL internet links provided to residential sites by telephone and cable companies. The content for the streaming media is presumed to be public domain and does not require protection. The overall functionality can be thought of as a decentralized p2p version of video streaming services such as YouTube or Revver[39][42].

3.1 Operation

DistVid sits on top of the Chord substrate depending upon it as the decentralized lookup mechanism. Any node wishing to participate on the DistVid network must already be a member of the Chord network and have been assigned a Chord id. A method is is assumed to be in place where by a DistVid node can discover the properties of any video that will be streamed on the network. This can be something simple like a meta file hosted on a web site. The relevant information consists of the name of the video stream, the number of segments in which the video has been divided and the number

of groups of pictures (GOPs) in which the segments have been divided. The contents of the meta file is a direct description of how the video stream is divided in order for DistVid to treat it as a "stream & file-share" hybrid. Figure 3.1 illustrates how a video stream would be divided.

The video "segment" is the smallest unit of video that will be shared amongst DistVid nodes. Each segment is further divided into a number of smaller units akin to bittorrent's pieces. Assuming a typical MPEG style video encoding scheme, these smaller units consists of a group of pictures (GOP) of a predetermined number of I, B or P frames[18][19]. The only caveat is that each GOP must be independently decodable. In other words, the very first frame of each GOP must be an I-frame.

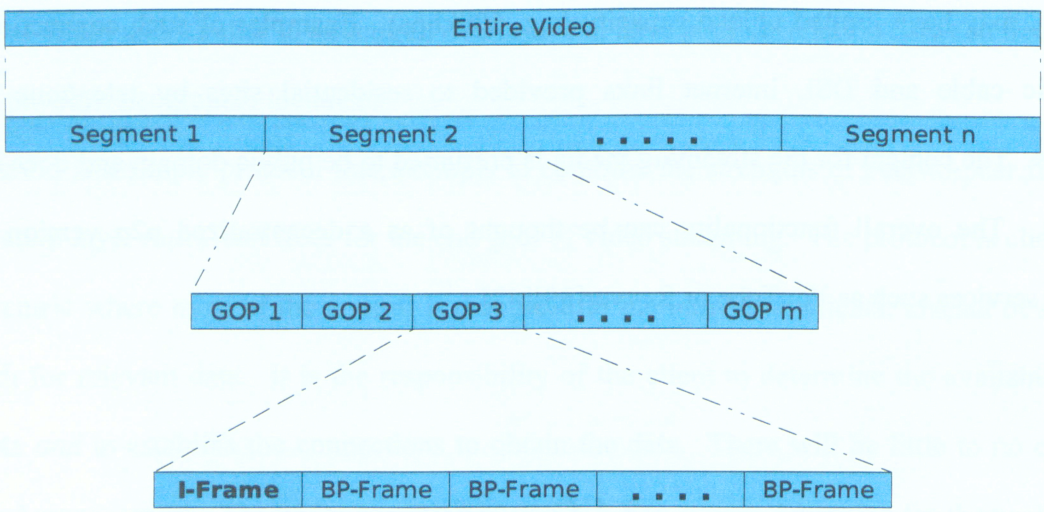


Figure 3.1 DistVid division of video stream

Decentralized tracking of video segments is accomplished as follows. The video's name followed by the segment number is hashed using the Chord key generating algorithm. This is commonly the SHA-1 hashing algorithm. The member on chord with the identifier that is the successor for the hashed value then becomes the node responsible for tracking nodes that have this particular segment of video. A node that has a complete segment(s) notifies all other nodes of what it has using a DECLARE message.

A DECLARE message consists of the following information. A hash of the video name concatenated with the segment number, the video's name and segment number, the node's own IP address and a time out value. This message is then routed via Chord to the responsible DistVid node.

The average node that participates on DistVid is assumed to have minimal storage space and thus will not hang on to any particular segment of video for an extended period of time. The time out value of the DECLARE message is used by the receiving Chord node to age its routing table of known segments. It is the responsibility of the DistVid node to announce to the Chord node the segment of video it is currently viewing and any other segments that will be held onto for any extended period of time with periodic DECLARE messages.

A DECLARE message arriving at a DistVid node is first checked for its hashed key value against the node's own Chord id. If the current node determines that it is responsible for that key the message is delivered to the DistVid application. Otherwise, the message is routed via Chord to the closest known node whose ID is the predecessor for the key value.

A DistVid node is responsible for tracking all peers that have video segments that map to the keys that it is responsible for. It accomplishes this by listening to the declarations by other nodes as they download particular segments of the video. As declarations arrive routing tables are built for each respective key. Because declarations are made to the Chord overlay rather than being broadcast to the entire network, messages arrive bounded by $O(\log_2 n)$ overlay hops. Figure 3.2 illustrates the activity of 3 DistVid nodes as they recover segments of video.

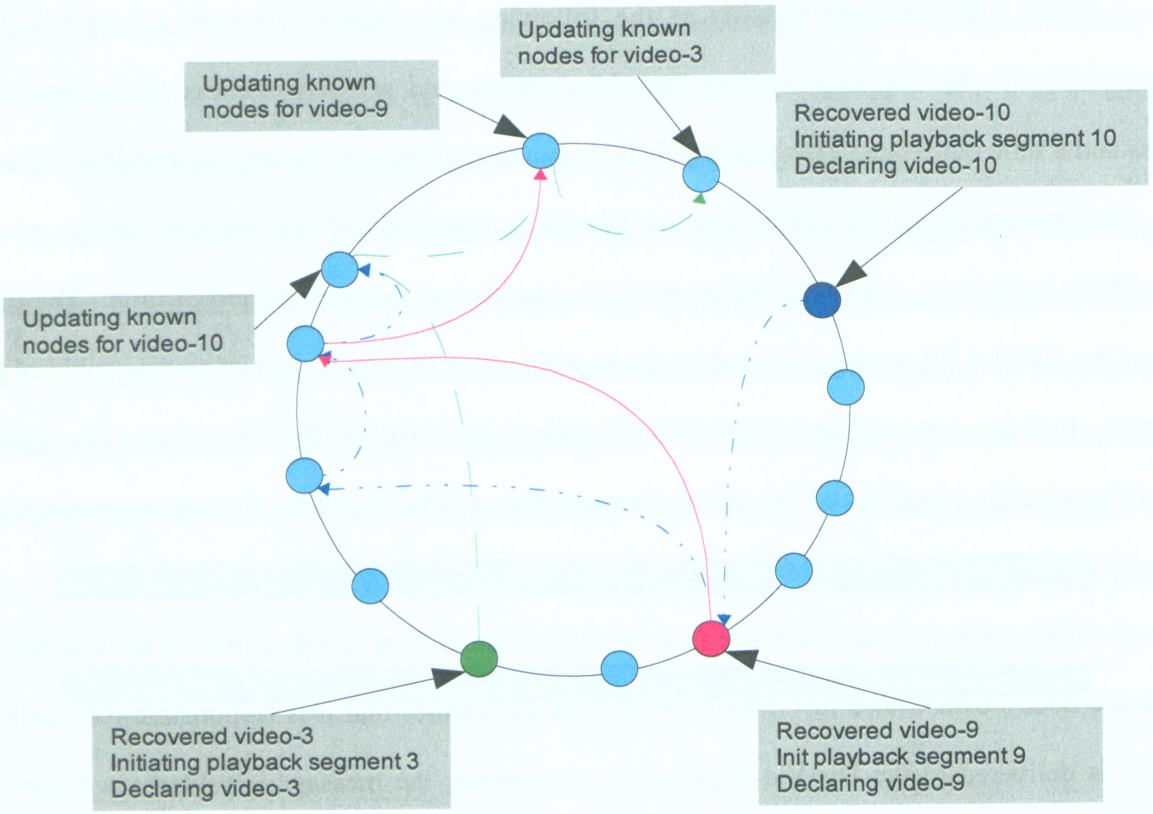


Figure 3.2 DistVid video declaration and tracking

A DistVid node wishing to join into a video stream first must determine the segment that contains the time of the video it wishes to view from the information provided in the meta-file. Having determined the segment, it generates the associated Chord key for that segment. A LOOKUP message is then routed over the Chord overlay to the node responsible for the target key. A LOOKUP consists of the requesting nodes IP address, the hashed key, the video name and segment number of the desired video segment. The responsible node then responds by sending back directly to the requesting node the IP numbers of a random subset of the nodes known to have the sought after video segment. Figure 3.3 illustrates this.

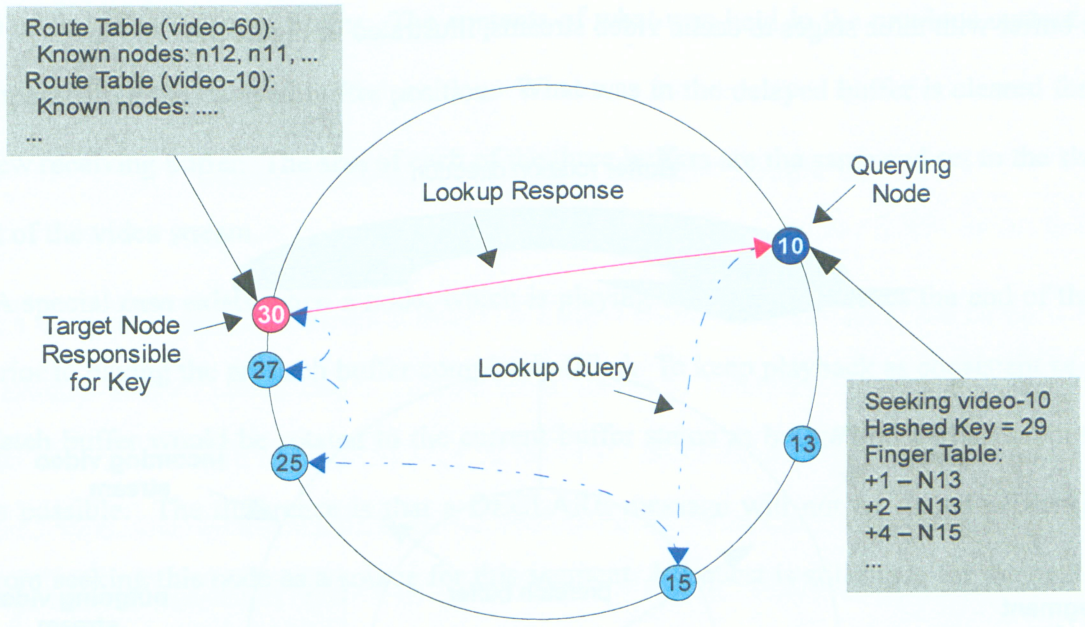


Figure 3.3 DistVid lookup

Note that should a node fail or leave the overlay unexpectedly, the lookup messages would be delivered to the node's successor in the Chord overlay ring. The response to the lookup would then be an empty list for that particular segment of video. Because all DistVid nodes periodically declare the segments that they have, the routing tables will eventually be repopulated at the new successor. Lookup operations would then continue normally despite the failed node.

Having received a list of nodes with the desired segment, the joining node randomly picks from the list of known nodes and tries to form a child/parent relationship. Nodes with which relationships are rejected by the potential parent are forgotten. Non-sampled nodes are remembered for future reference in case another parent needs to be found. The rationale being that the node performing the lookup and potential source nodes are approximately in the same time frame and are thus moving through the video stream at approximately the same positions. Thus they would potentially make good parents should another parent need to be found at some future time.

Acting as a parent and a child, a DistVid node in its most minimalist configuration implements a

ring like buffer with three stages to cache video streams, illustrated in Figure 3.4.

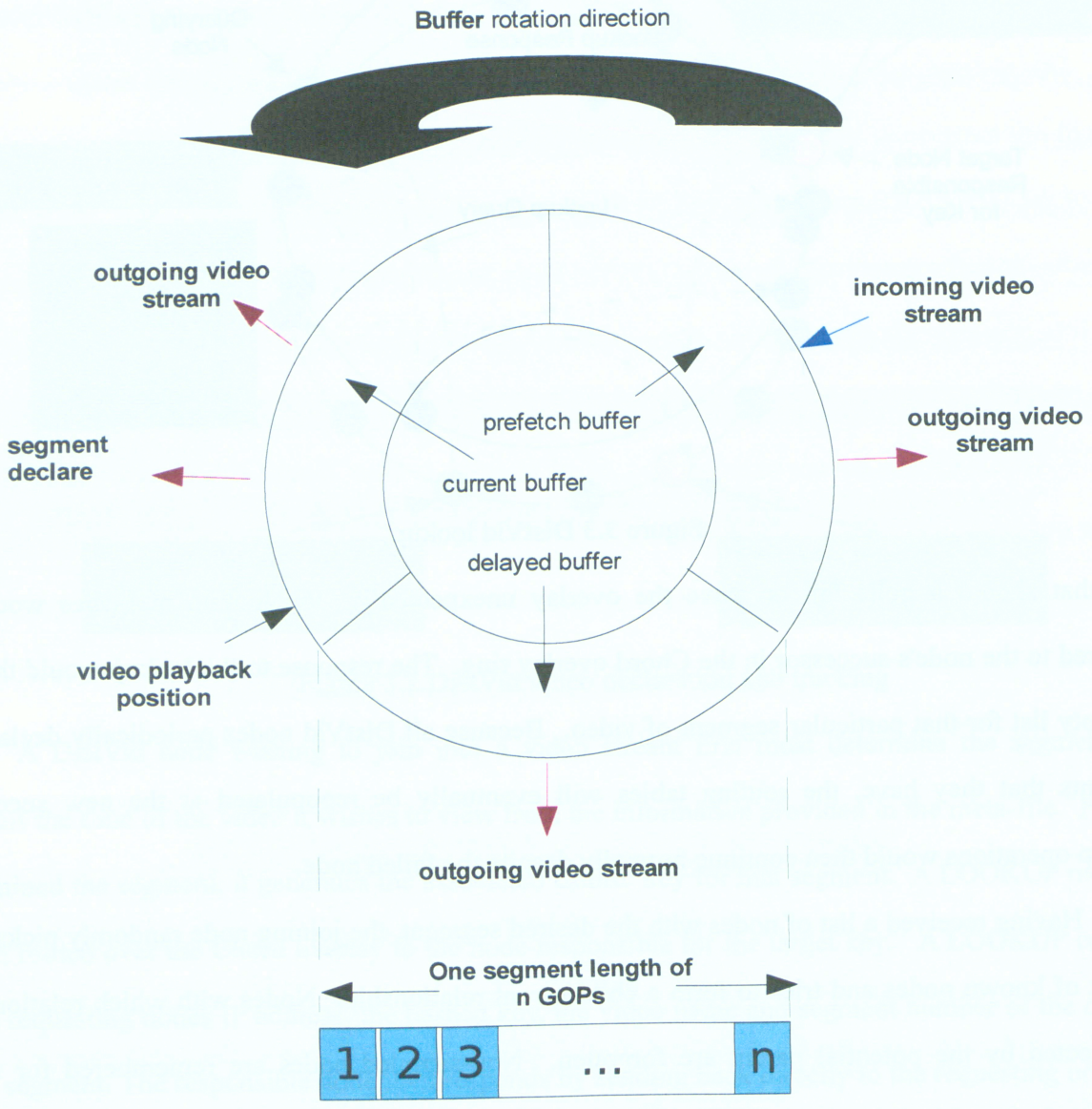


Figure 3.4 DistVid video buffering

The prefetch buffer is used to store in advance of where video playback is taking place. When the prefetch buffer is full and the client is ready to begin playback, the ring is rotated such that the prefetch takes on the current buffer position for playback. At this time, the DistVid node will issue a DECLARE message, routed through the Chord overlay, to the tracking node informing it as to the

contents of the "new" current buffer. The contents of what was held in the previous current buffer is held temporarily in the delayed buffer position. What was in the delayed buffer is cleared for it to act as the new receiving buffer. The size of each of the three buffers are the same and set to the size of one segment of the video stream.

A special case exists when a node, which is playing back video, reaches the end of the current buffer prior to having the prefetch buffer completely filled. To keep playback as consistent as possible, the prefetch buffer would be rotated to the current buffer status as before and playback continued as much as possible. The difference is that a DECLARE message will not be issued preventing other nodes from seeking this node as a source for this segment. A request is still made for the next segment to fill the new prefetch buffer while any late arriving GOPs for the incomplete segment will be used in the current buffer if playback has not reached that particular GOPs position yet.

Some may question the necessity of the delayed buffer, however, its purpose can easily be seen with a simple use case. Suppose a node has declared its current buffer and has begun playback. In fact, playback has progressed to a point near the end of the current segment. At this time, a child node requests the current segment from this particular node. As the current node transitions to the contents of the receiving buffer, that is the prefetch buffer is rotated to "current" status, there has been insufficient time to fully stream the current buffer out to the node's children. Thus, a parent node needs to hang on to an expired current buffer as a delayed buffer for, in the worst case, the entire length of the video segment.

3.2 Parent-Child Relationship Formation

A child-parent relationship is formed when a child sends a parent request message to a potential parent requesting a particular segment of video. A REQUEST is sent directly via IP address and not routed through Chord. Two methods exists for selecting parents. Closest neighbor via link latencies / Vivaldi co-ordinates or else randomly. As mentioned earlier, recent implementations of Chord have incorporated Vivaldi co-ordinates, however, because the necessary modifications have not been implemented to forward Vivaldi information to the application level. the work presented here are the result of random selection similar to the method suggested by Zhang et. al.[41].

A node receiving a parent request message checks to see if the requested segment is indeed available, the number of children that it has has not exceeded a predetermined maximum, and that the requesting node is not already a child of this parent. Meeting all three of these criteria, the parent node will add the requesting child to its child queue. In the case where the requesting node is already a child on queue, the parent will take the REQUEST message as both a keep alive message and as a confirmation that the child is ready for the next segment. If a parent cannot accept the requesting node as a child a REJECT message is sent back.

A DistVid parent can be set to rank its children, based on their reported tree weight, and to enable a priority queue. The tree weight of a child is the total number of nodes that are directly and indirectly dependent on the child for a particular video stream. The higher the tree weight the higher the child is ranked. Based on a preset priority queue size n, a parent will service the top ranked n children in a round robin fashion before the rest of the children are serviced provided sufficient

bandwidth is available.

Parent nodes service their children by forwarding available GOPs from the requested segment, as quickly as possible, to the child node. Forwarding of GOPs within a segment is automatic with the delivered GOPs taking on a secondary role as a keep alive message. A child that does not hear from a parent in a duration greater than one segment length automatically assumes the parent has either left the network or is no longer a good source for the video stream. It is important to note that DistVid makes no guarantees as to the order in which GOPs within a segment are forwarded. While potentially sequential, GOPs will very likely be chosen ***randomly*** from within the desired segment. Indeed, when multi-parenting is discussed in section 3.5, random delivery is actually preferred. It is up to the child to reorder the arriving GOPs before playback. It is important to note that this is not the random packet arrival inherent in UDP packet transmission but the intentional randomization of GOP transmission at the application level.

A child node on receiving a REJECT message will forget the rejecting parent and attempt to contact another node on its queue of potential parents. The arrival of video streams, in the form of sent GOPs, will be taken as acknowledgment of the child-parent relationship by the parent. The child at that point will quietly await for remaining GOPs of the segment to arrive. Arriving GOPs serve a secondary role as a keep-alive message as well.

When a child's receive buffer is full and it begins its playback, the child node announces via a DECLARE message that it has the desired segment and begins playback of the video. A request is then made to the parent for the next segment. The same procedure is applied when the node's current playback position reaches the end of the current buffer yet the prefetch buffer has not been completely filled. The prefetch is as before rotated to the current position and a request made for the next segment. However, in this case, a DECLARE message is ***not*** issued.

3.3 Breaking of the Parent-Child Relationship

A DistVid node in a child-parent relationship, can at any time, try to move to another part of the tree, providing no guarantees to its peers. Examples of such situations include a node performing fast forward or rewind functionality or else a node may decide that it's parent is not streaming the necessary video in a sufficiently reliable manner. In such cases, it is the responsibility of the children to locate another parent.

A parent child relationship breaks under numerous conditions. Recall that a REQUEST message serves also as a keep alive message from the child. If a parent has not heard from the child over the duration of one segment, it will assumes the child has failed and removes it from its service queue.

A parent may also leaves the video stream or decides to seek to another position in the video stream that requires it to jump to another segment. A DISCONNECT message is issued to all active children. This message informs the child that while the parent will continue to stream available content, the child should seek another parent as soon as possible.

Just as a REQUEST message serves as a keep-alive message from a child, arriving GOPs plays a secondary role as a keep-alive message from the parent. A child who has not received a GOP over the duration of one segment will assume that contact has been lost with its parent and will seek to re-parent with another node.

Other factors that may induce a child to leave its parent is determined by the conditions of the prefetch buffer. A child may also intentionally leave an active parent if it deems it an unsuitable source for video. DistVid sets two values , *LOWER_PREFETCH_FILL_THRESHOLD_PERCENT* (e.g. 60%) to define a minimal acceptable parent and UPPER_*PREFETCH_FILL_THRESHOLD_PERCENT* (e.g. 90%) above which is considered a good parent. The first value determines the lowest percentage the prefetch buffer is filled in the duration of one segment time that the child will accept. The percentage is calculated based upon the number of GOPs received out of the total number GOPs in the segment. If the prefetch buffer is filled below this value the child will decide that this particular parent is an unsuitable source and will seek another parent. The second value is used to track how frequently the prefetch buffer is filled below the upper threshold and above the lower threshold. If the prefetch falls between this range more than a predetermined number of times (e.g 3) the DistVid node will perform a random coin toss to decide if it should seek another parent.

3.4 Queuing Strategies

While the sustainability of a parent child relationship in DistVid is predominantly decided by the child, the amount of time required for a child to decide a parent is unsuitable can be substantial (2-3 segment lengths). During this time the child may be receiving very poor media quality. As well, frequent re-parenting activities by a particular node can have detrimental effects on the parent child relationships of other nodes. To this end, as mentioned earlier, a DistVid node acting as a parent, can establish a priority queue for its children based on its own perceived upload capacity. By selectively starving particular children, a parent can encourage its child to decide earlier on that the existing parent

child relationship is unsuitable while at the same time protecting nodes that have numerous other dependents upon it. A priority queue will also act to protect more important children from the disruptive influence of other nodes joining and leaving. This thesis examines DistVid using setups with no priority queue (***nopq***), limiting the number of nodes to match the upload capacity (***capped***), a priority queue that maximizes upload capacity (***pqmax***), and a priority queue that is at 67% of upload capacity (***pq67***). Both pqmax and pq67 strategies accept a total number of children greater than their queue sizes and, for the simulations presented here, equal to doubling of the capped value.

The priority queue employed by DistVid is based upon the number of nodes dependent on a particular child. For the single tree configuration, this will be the weight of the sub-tree (number of nodes) under a given child. Sub-tree weight values are conveyed by a child to its parent either through proactive update messages sent by the child whenever it notices the topology has changed or else by piggybacking the weight value on segment request messages. Proactive updates allow the overlay topology to react quicker to changes at the expense of increased network traffic. The experimental results presented in this thesis use the proactive approach. The Table 3.1 outlines the operation of each type of queue.

nopq – No priority queue and number of children is unlimited	• A node will treat all children on its queue fairly and will, in a round robin fashion, send exactly ***one*** GOP to each child in its queue before sending any more to a given child. The node will also accept any number of children provided the child is seeking content that this particular node has. Children are serviced in a round robin fashion with the smallest service unit being the transmission of one GOP.
capped – Number of children a node accepts is limited.	• Similar to nopq except a node will not accept more children than its upload bandwidth can theoretically send to (based upon GOP size). Children are serviced in a round robin fashion with the smallest service unit being the transmission of one GOP.
pqmax – Priority queue the size of capped value.	• A node establishes a priority queue to the size of the number of nodes it can theoretically handle (like capped). Pqmax, however, allows more nodes to enter its child queue but will will service only the nodes in the priority queue first so long as the nodes in the priority queue have requests. Nodes outside the priority queue will receive GOPs only after all nodes within the priority queue have been serviced to completion. Priority is based upon the child's weight which is the number of nodes that is directly or indirectly dependent on that child. • All children within the priority queue are serviced equally. • All children outside the priority queue are also treated equally. • The total number of children a node will accept is twice the capped value. • Older children (i.e. nodes that have been a child to a node for a longer period of time), are prioritized over younger children when their weights are the same.
pq67 – Priority queue at 67% the size of the capped value.	• Like pqmax, however the priority queue size is intentionally limited to 67% of the theoretical maximum.

Table 3.1 Queuing strategies

3.5 Necessity of Queuing Strategies

In this section the necessity of queuing strategies on the part of the parent will be discussed. As mentioned earlier, the priority queue is an effective way to encourage children to leave when a particular parent is heavily loaded. However, since network conditions continually change depending on congestion, it may be impossible to reliably predict a good maximum children value for strategies such as capped and pqmax. As will be shown in chapter 4, utilizing a priority queue that underestimates the capped value can provide performance that rivals if not betters a capped strategy.

There are other reasons that that makes a priority queue particularly useful. Figure 3.5 illustrates one such case.

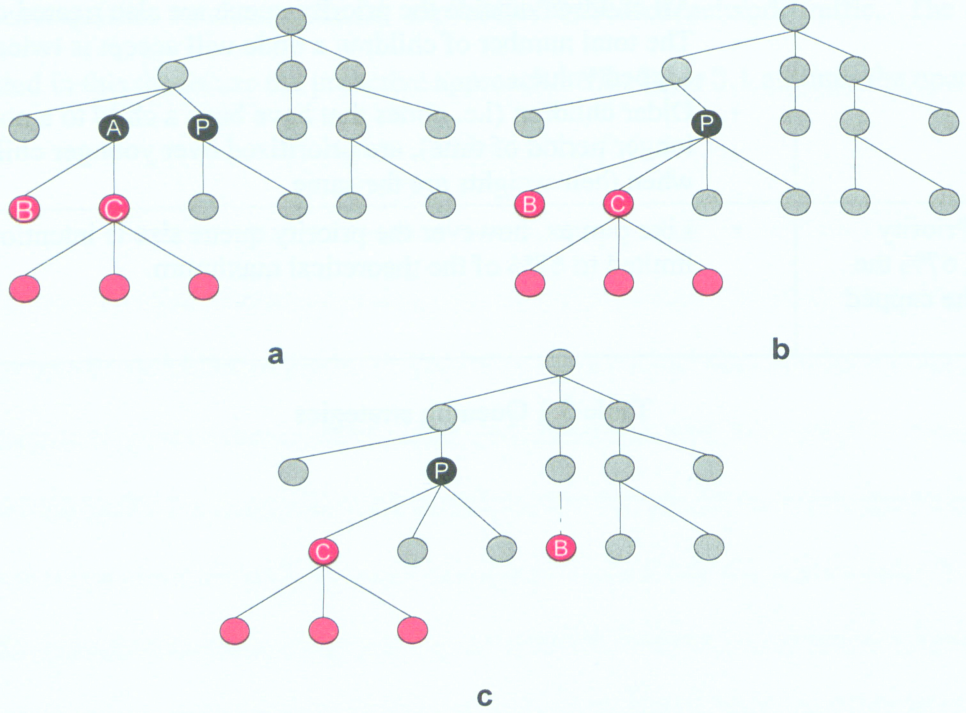


Figure 3.5 Effects of priority queues on node leaving

In the above scenario, suppose each node can support 3 children each and node A decides to leave (a). A detaches from its parent P and nodes B and C become aware of A's leaving either through a DISCONNECT message or time out. Suppose P is now a good parent for B and C and both attempt to form a relationship with P (b). In the capped scenario, if B becomes P's child first, C and its children will have no choice but to find another suitable parent. However, with a queuing strategy, irregardless of who joins with P first, C is prioritized because of the number of children that are dependent upon it (c). Thus, in this scenario, a priority queue acts to minimize disruption to the most number of nodes by letting parent nodes favour nodes with more dependents. B being a non-prioritized node will likely receive poor service causing it to eventually seek another parent.

In section 4.3, the benefits of a priority queue over a capped strategy is further demonstrated through simulation results.

3.6 Multiple Parents

DistVid is geared to an environment where nodes have asymmetrical links with high download bandwidth but reduced upload bandwidth such as a 2Mbps download / 600kb upload connection. In this environment, the goal is to maximize video quality as well as network bandwidth utilization. DistVid can achieve this by leveraging the strengths of bittorent style swarm downloading. However, to do this, one must divide video in such a way that multiple smaller independent units can be recovered from different parents to give a much higher quality aggregate whole. However, the division of video, as we improve video quality through increased bit rate, is no simple task in itself. Bittorrent style division such as that employed in the single-parent model may be insufficient.

Recall that the video is to be divided into GOPs as the smallest transferable unit, similar to bittorrent's "piece" concept. Keeping with this model, if one was to increase the quality of the video stream, one must then increase the bit-rate at which the frames within the GOP are encoded. For example, suppose we maxed out our upload capacity for this scenario and transmitted a 600kbps video stream using the single parent model. A one second GOP alone will consume the entire upload bandwidth of a client node destroying the network's ability to support a tree style broadcast.

One may then argue to make to pieces "smaller" by reducing the number of frames per GOP and then increasing the number of GOPs per segment. However, neither is this a viable solution. Consider a traditional MPEG4 **[40]** style encoding scheme and recall that each GOP needs to be independent. This dictates that the first frame of each GOP be an I-frame. Considering that the I-frame

is the single largest consumer of data-bits in a group of I, B, and P frames, then to even decrease a GOP length from 30 frames to 10 frames will have a large negative effect on the compressibility of the video by increasing the total number of I frames.

After a video has been divided into GOPs, a method needs to be in place to transmit the individual GOPs from multiple peers. One method, using traditional video encoding, is to form relationships with multiple parents having the same segment of our desired video. The child then coordinates with its parents to determine who is to send what. For example, for two parents, a child may request one parent to send every odd GOP of a segment and the other every even GOP. However, as one may surmise, as the number of parents grows, coordination can get quite complex.

An alternative to the method outlined above, and what is examined in this work, is to use multiple description encoding. Multiple description coding, as explained in Chapter 1, is a well established encoding style that allows a video to be broken down into multiple descriptions. MDC encoding offers two important advantages. First, each description, like a whole video stream in itself, is independently decodable. A node which recovers a single description can essentially watch the entire video albeit at a reduced quality. The second advantage is that descriptions can be combined to increase the overall aggregate video bandwidth and thus quality.

Taking advantage of the benefits offered by MDC encoding, DistVid employs a bittorent style swarm download to maximize the video quality recovered by its nodes. In the sections to follow, DistVid will be studied with a simplified MDC model whereby the total aggregate decoded video bandwidth is the sum of the bit rates of each individual GOP irregardless of which description it belongs to.

DistVid uses the same the same DECLARATION/LOOKUP strategy under multiple-parenting as it did in the single-parent single video stream model. Indeed, each description produced under MDC encoding is in itself treated as an independent single video stream with a single parent. A DistVid node

will attempt to form multiple peer relationships with nodes that have different descriptors of the same segment of video. The overall effect is of a node joining multiple multi-cast tree/mesh type networks in the attempt to recover as many descriptors as possible. Each mesh is then streaming an independent descriptor of the desired video. Figure 3.6 illustrates this.

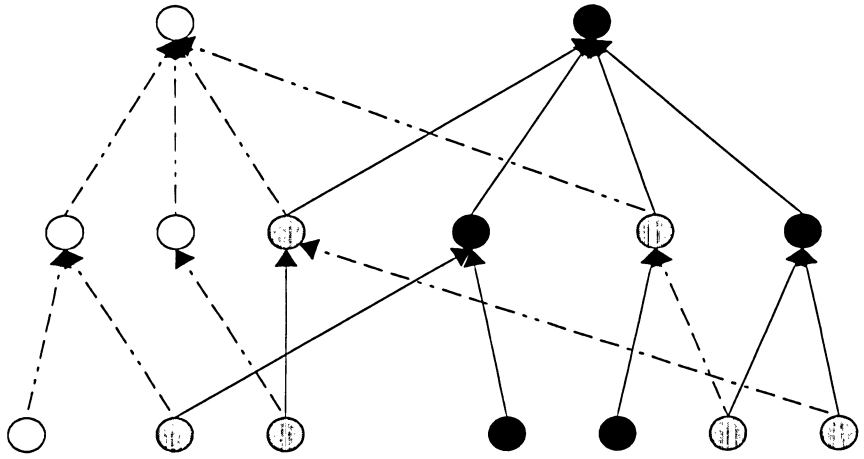


Figure 3.6 DistVid node multi-parent relationships

A child is attempting to recover three different descriptions of a particular segment of video. It thus forms relationships with parents 1, 2, and 3. Note that each of these parents may themselves have more than one particular description of the video. However, with its relationship with this particular child parents 1, 2, and 3 are responsible only for streams A, B, and C respectively. To maximize the *chances of the child recovering most parts of the desired segment, a parent does not send out GOPs of a* segment sequentially. Instead, a parent will forward GOPs from the segment in a random fashion. Thus, for a child that is unable to recover all GOPs for a given segment of a particular description, it may be able to recover all GOPs for a segment from at least one of the three descriptors allowing continuous playback.

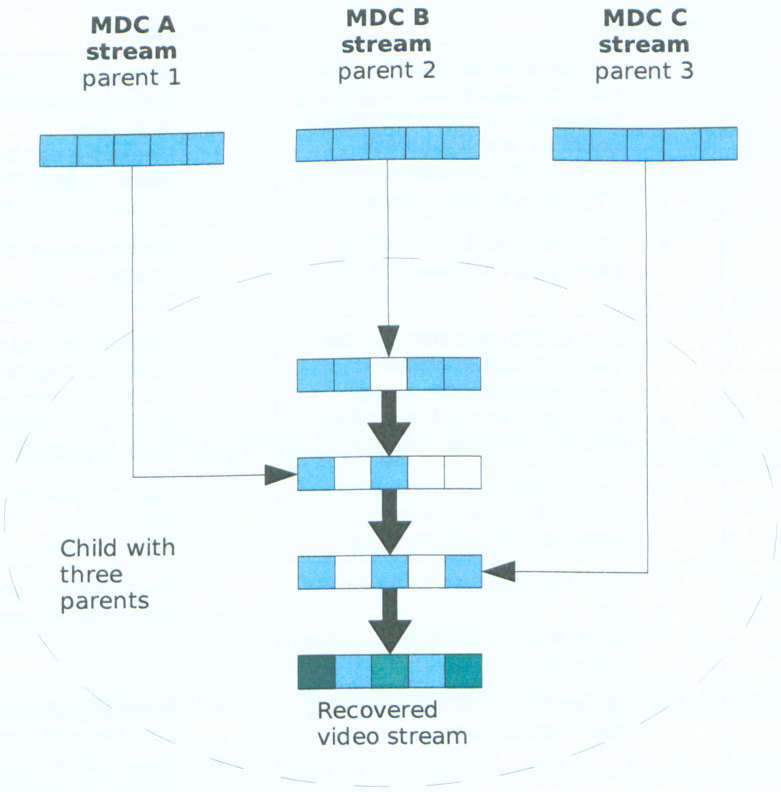


Figure 3.7 Multi-parent swarm style video streaming

3.7 DistVid Messages

DistVid has only a small number of communication messages that is implemented over Chord. Table 3.2 summarizes them.

Message Name	Contents	Comments
GenericMessage	• sourceNodeHandle: PlanetSim node handle of node originating message. Includes the node's chord ID, and IP address • messageType: integer identifier • TTL: Time to live in hops	• **ALL** DistVid messages subclasses GenericMessage and thus have theses properties. • Messages can be sent ***overlay point-to-point*** via their IP addresses or ***routed*** in a decentralized manner.
ChildDisconnect	• streamName: Unique string identifying the video stream	• Notifies parent that the sending nodes is leaving. • Type: point-to-point
DeclareMessage	• key: Chord algorithm hashed value of <streamName>-<segmentNumber> string. • ttl_steps: Time to live for this new route in seconds	• Used to declare a node as a source for a particular segment of a video stream. • For the implementation on PlanetSim time to live is in simulations steps. • Type: routed
LookupMessage	• key: Chord algorithm hashed value of <streamName>-<segmenetNumber> string.	• Used to retrieve sources of a particular key. • Type: routed
LookupResponseMessage	• key: Key that was being looked up • data: array of IP values of known sources	• Response to LookupMessage • Type: point-to-point
ParentRejectMessage	• streamName: String identifying the video stream	• Receiving child stops requesting stream from sending node. • Breaks an existing relationship or else refuses a request to form a relationship • Type: point-to-point
ParentRequestMessage	• title: streamName • segmentNumber • gopAlreadyHaveMap: bitMap of GOPs already received • treeWeight: total number of nodes in sending node's sub-tree • treeDepth: maximum hight of sending node's sub-tree	• Requests a segment number from a particular stream from a parent. • Serves also as a keep-alive message and an update of the known sub-tree weight (total number of nodes) and depth (total height of subtree). • Type: point-to-point
ReportTreeWeight	• weight • depth • streamName	• Used for proactive updates of changes in the known sub-tree weight and depth • Type: point-to-point
VideoStreamMessage	• title: streamName • segmentNumber • gopNumber • gopData: binary data	• Transmission of one GOP • Type: point-to-point

Table 3.2 DistVid Messages

Chapter 4 DistVid Single Parent Evaluation

DistVid has been implemented on PlanetSim with the intention of eventually testing on PlanetLab to obtain real world data. PlanetSim is step driven and has been configured to execute one third of a second per step under the following assumptions:

1. Upload video bandwidth of non source node: 600kbps
2. Download video bandwidth of non source node: 2000kbps
3. Upload video bandwidth of source node: 1200kbps
4. Video: 200kbps CBR encoded with no MDC or layer coding used
5. The existing network topology has the capacity to deliver up to 100 messages per node per second.
6. Leave threshold: <60%
7. Good parent threshold: >90%

As mentioned in section 1.8, PlanetSim does not model an actual packet network. To approximate some of these effects, we have limited node to node traffic to a maximum of 33 messages per step (DistVid and Chord traffic inclusive). As well, because the predominant consumer of bandwidth in our tests will be video traffic, video traffic has been limited three 200kbit GOPs per three simulation steps. Thus PlanetSim provides a platform approximating a network where on average, each node can send/receive ~99 message per second (3 steps) with a peak video upload bandwidth of

600kbps of video.

Multiple criteria were used to assess the performance of DistVid. These include, the stability of the topology, the effects of the size of a video segment, and the ability of nodes to recover video bandwidth. While not intended to be exhaustive, many simulation hours were run. The results presented in the following sections were selected because they appeared to represent the generalized result of the simulations. For this chapter, nodes in these simulations attempt to find only one parent. To seek another parent, a node will first abandon any relationship with an existing parent. Note also that the simulations assumes that participating nodes have already joined the Chord overlay and that the overlay topology has stabilized.

In part 4.1, we shall examine the DistVid topologies that are formed from the various queuing strategies, mentioned earlier, on a small 30 node configuration where 20 nodes are actively participating in the video stream. In part 4.2, the stability of the topologies are examined on a 600 node configuration where 500 nodes are actively participating in the video stream. The metrics used will be the number of nodes without parents at a given moment of time, the number of attempts to re-parent and ultimately the received video bandwidth per node. Finally, section 4.3 examines the effect of varying segment sizes.

For sections 4.1 and 4.2, simulations were run with a video stream divided into 120 segments with each segment being 30 GOPs long. Each GOP was assumed to be 1 second long consisting of 30 frames. There is only one source node that has the entire video and each node participating in the video stream try to recover the video from time 0.

4.1 Video Stream Topology

4.1.1 Capped with no priority queue *(capped)*

A conservative approach, this queue strategy restricts the number of children a DistVid node is allowed to take on stay at (or just below) the node's streaming capacity (available bandwidth). Nodes are set to attempt to join the video stream randomly with high probability (many joining early on and few joining later on) . The maximum number of children a node is allowed to take on is determined by dividing the upload bandwidth by the video bit-rate and taking the floor of that value. A round-robin scheduling is implemented for all children.

Figure 4.1 illustrates the topology that forms from an example run of a network consisting of thirty nodes where twenty are trying to participate in the video stream from the beginning.
Observe that the topology stabilizes quickly and is essentially unchanged after 120 seconds. Nodes find good parents quickly. However, note that a fair number of nodes, especially those joining later in time, are never able to find suitable/stable parents.

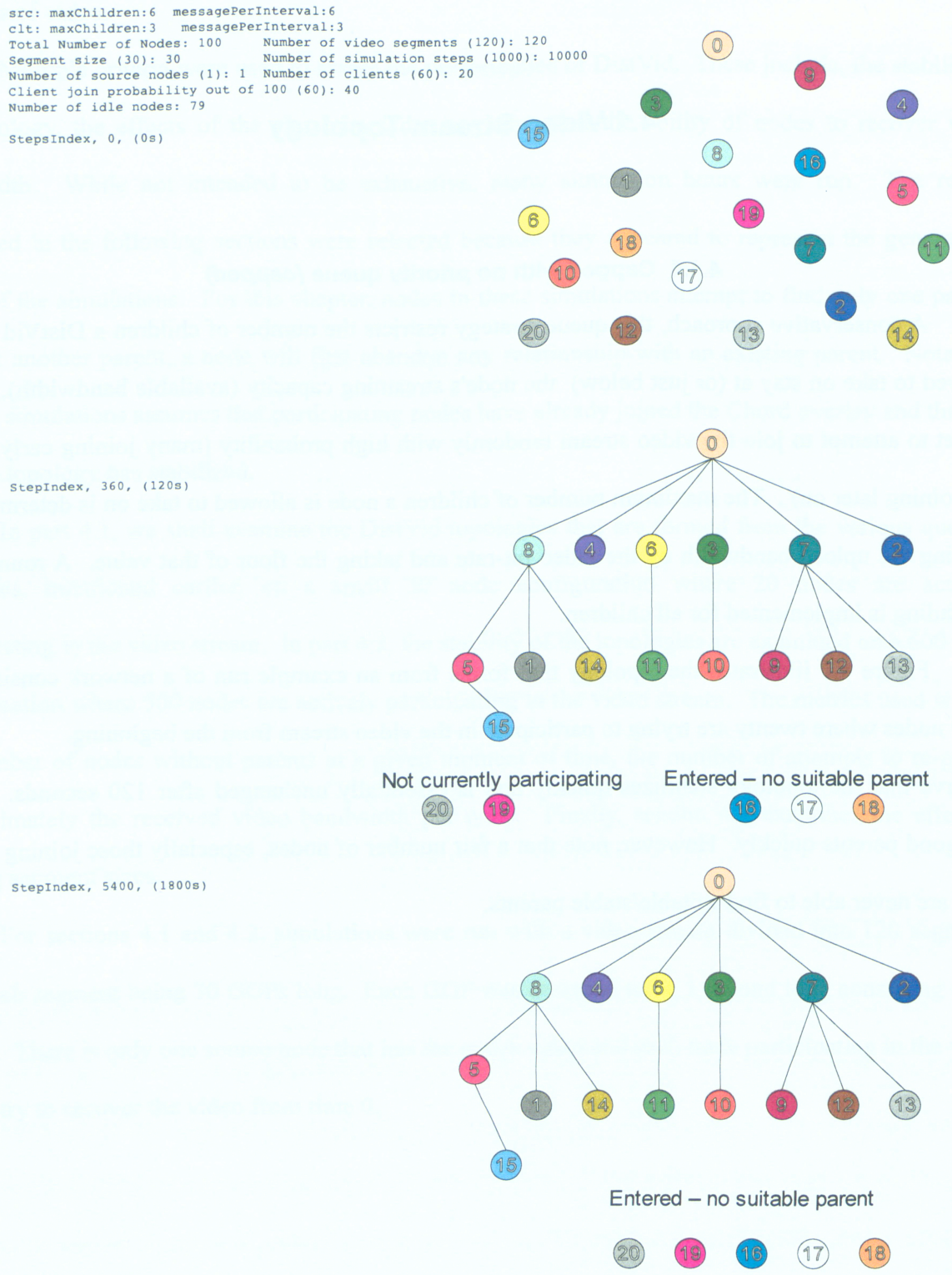


Figure 4.1 DistVid Topology (Capped)

4.1.2 No priority queue *(nopq)*

Network congestion is simulated by allowing a node to take on enough children to overwhelm its upload bandwidth. Here the source node, is allowed to take ten children each requesting a 200kbps data stream. A child node is allowed five children. A DistVid node keeps count of the number of times the receiving buffer has failed to completely fill above a predefined threshold *(UPPER_PREFETCH_FILL_THREHOLD_PERCENT)*. If the count exceeds the predefined maximum, the node will perform a random coin toss to determine if it should find another parent. As well, if a child fails to receive, after the duration of one segment, a threshold percentage of the segment *(LOWER_PREFETCH_FILL_THRESHOLD_PERCENT)* it will assume the parent is not a viable parent and will voluntarily leave. As before, round-robin scheduling is used for all children.

Figure 4.2 illustrates the resultant topology of an example run. Observe that connections form and break rapidly indicating poor connection quality which result in nodes frequently seeking better parents. Because no preferential treatment is applied to any of a parent's children, a new node attaching to a parent will affects all children of that parent by consuming upload bandwidth. Thus the action of a single node can cause a large number of nodes to suddenly re-parent resulting in large disruptive changes in the topology.

This is especially true if a node that is influenced to move is the root of a very large sub tree. Should a new parent not be recovered quickly enough, all nodes in the sub-tree would be affected potentially causing a large number of them to seek out new parents. As can be seen from the following figure, the DistVid topology never quite stabilizes changing rapidly throughout the simulation. While there were periods where the tree became relatively stable, the movement of a single node had drastic destabilizing effects.

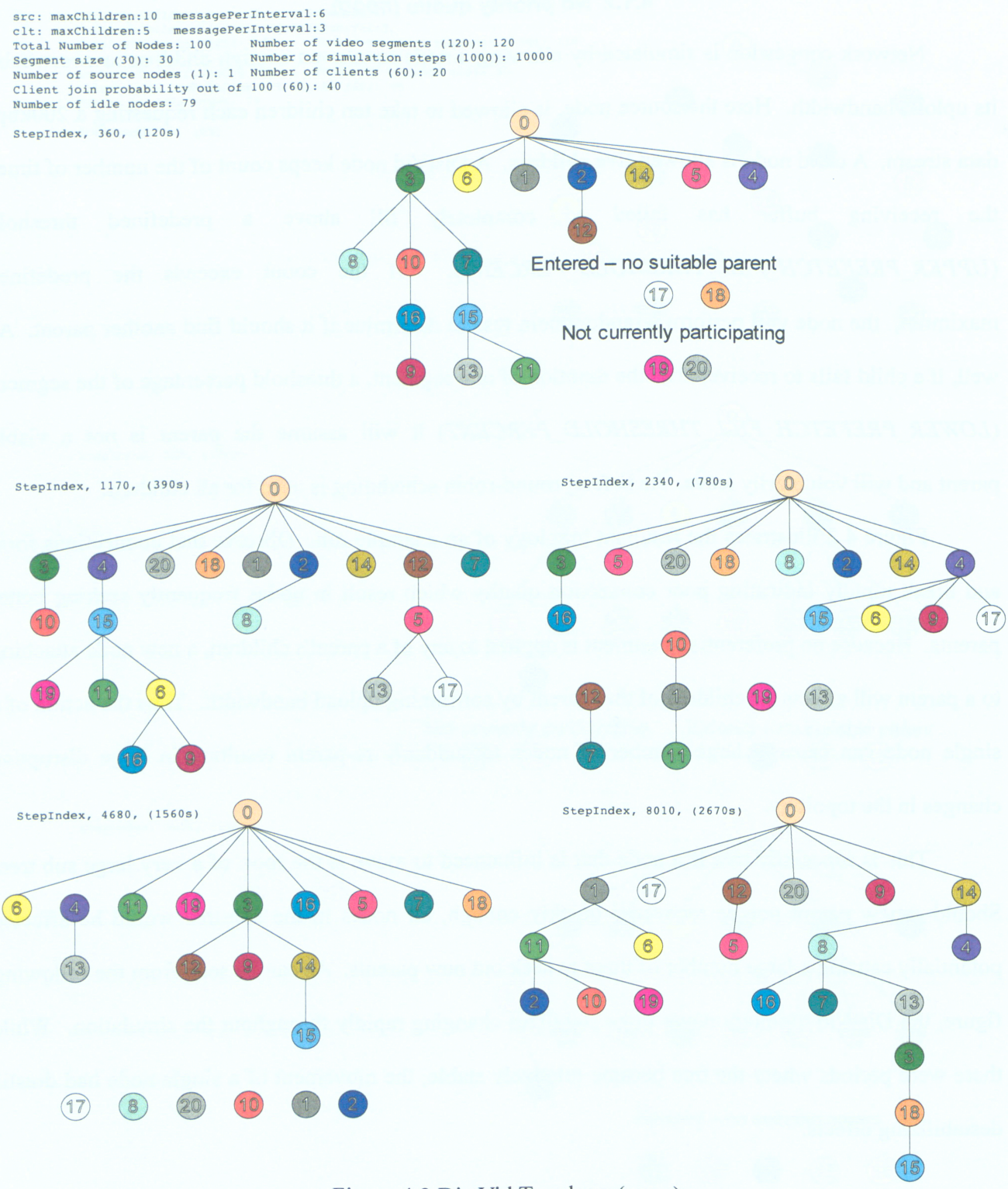


Figure 4.2 DistVid Topology (nopq)

4.1.3 Maximal priority queue with selective starvation (*pqmax*)

In this section, DistVid nodes, based on available outbound bandwidth was set to prioritize their children. Those with the largest number of nodes in their subtrees were preferred, guaranteeing service to those nodes. The remaining children outside the priority queue would be serviced only when bandwidth was available. The size of the priority queue is maximized by setting it to the same value as the *capped* policy discussed in section 4.1.1. At the same time, nodes were allowed to take on children in excess of their upload bandwidth. In this case the source node was allowed to take on ten children with a priority queue size of six. Similarly, non-source nodes were allowed to take on five children with a priority queue size of three.

From figure 4.3, we can see for this particular run of the simulation, the topology stabilizes quickly compared to the run where neither capping or priority queues were used. For the majority of the nodes, the topology is essentially unchanged beyond 360 seconds. Because of the queuing strategy employed, large stable sub-trees were able to form even as new nodes entered the video stream.

A particularly interesting observation is the number of children that are able to form stable relationships with their parents. While we would expect the number of children greater than the priority queue size would be unstable it would appear that the topology does not favor a maximal queue size either. The source node which has a queue size of six appear more stable with five children nodes. Similarly, non-source nodes appear more comfortable with only two children although its queue size has been allocated to three nodes.

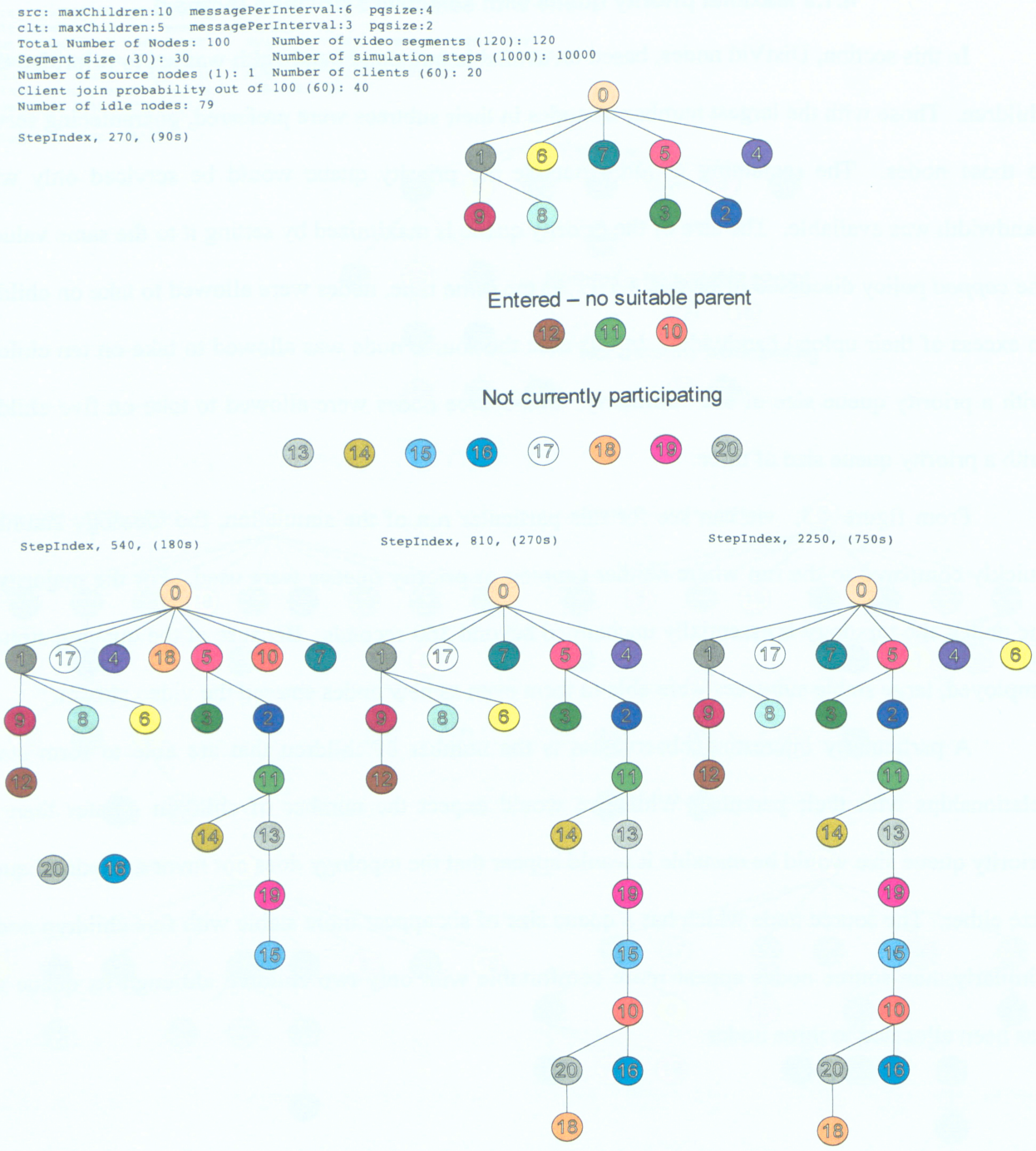


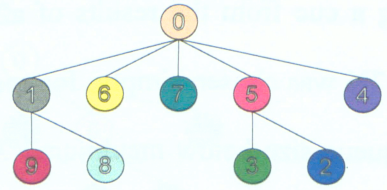
Figure 4.3 DistVid Topology (pqmax)

4.1.4 67% priority queue with selective starvation (pq67)

Taking a cue from the results of a2, the priority queue was reduced to two thirds (67%) of maximum. 67% was chosen simply for ease of implementation with the goal of examining the result of setting a queue size below maximum. As with *pqmax* the topology stabilizes a little slower than when *capped* but significantly faster than when no priority queue *(nopq)* was used at all. For the majority of the nodes, the topology is unchanged beyond 270 seconds with the system once again preferring large stable sub trees.

The benefits of *pq67* over *pqmax* is not readily apparent here. Section 4.2 of the evaluations more effectively demonstrates the advantages of using a smaller priority queue.

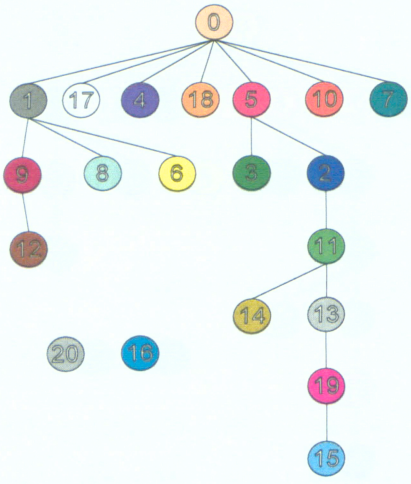
```
src: maxChildren:10  messagePerInterval:6  pqsize:4
clt: maxChildren:5   messagePerInterval:3  pqsize:2
Total Number of Nodes: 100      Number of video segments (120): 120
Segment size (30): 30           Number of simulation steps (1000): 10000
Number of source nodes (1): 1   Number of clients (60): 20
Client join probability out of 100 (60): 40
Number of idle nodes: 79

StepIndex, 270, (90s)
0 - 3739280700: ent:-1 p:null s:0 tWt:10 child:938809077(2)
2715473618(2) 1947209534(0) 358940373(0) 1451360013(0) 1053176866(0)  *
1 - 938809077: ent:90 p:3739280700 s:1 tWt:2 child:3661511084(0)
2054479009(0)  *
2 - 1773202509: ent:90 p:2715473618 s:0 tWt:0
3 - 2339950336: ent:90 p:2715473618 s:0 tWt:0
4 - 358940373: ent:90 p:3739280700 s:1 tWt:0
5 - 2715473618: ent:90 p:3739280700 s:1 tWt:2 child:1773202509(0)
2339950336(0)  *
6 - 1451360013: ent:90 p:3739280700 s:0 tWt:0
7 - 1947209534: ent:90 p:3739280700 s:1 tWt:0
8 - 2054479009: ent:90 p:938809077 s:0 tWt:0
9 - 3661511084: ent:90 p:938809077 s:0 tWt:0
10- 1053176866: ent:180 p:null s:0 tWt:0
11- 1971076203: ent:270 p:null s:0 tWt:0
12- 3069590560: ent:270 p:null s:0 tWt:0
13- 241493007: ent:-1 p:null s:-1 tWt:0
14- 188494369: ent:-1 p:null s:-1 tWt:0
15- 67976858: ent:-1 p:null s:-1 tWt:0
16- 1083175481: ent:-1 p:null s:-1 tWt:0
17- 1679150684: ent:-1 p:null s:-1 tWt:0
18- 883700810: ent:-1 p:null s:-1 tWt:0
19- 1656703157: ent:-1 p:null s:-1 tWt:0
20- 1081660188: ent:-1 p:null s:-1 tWt:0
```



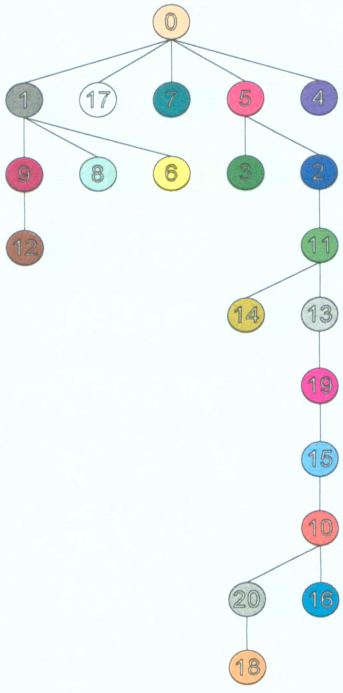
Entered – no suitable parent

Not currently participating

StepIndex, 540, (180s)



StepIndex, 810, (270s)



StepIndex, 2250, (750s)

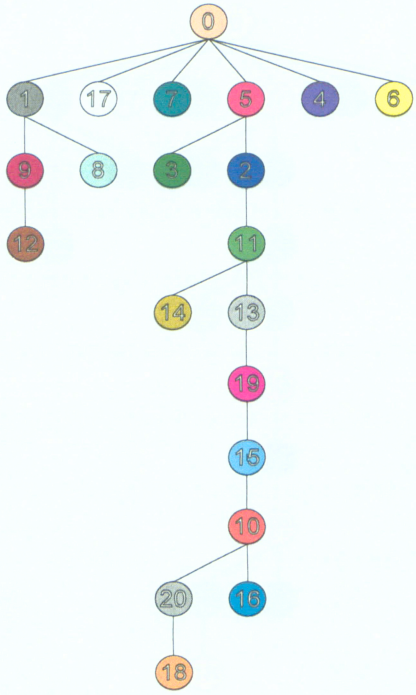


Figure 4.4 DistVid Topology (pq67)

4.2 Topology Stability

In the sections that follow, DistVid is examined on a much larger scale network. The following data were obtained using 600 Chord nodes where 500 nodes are actively participating in DistVid.

4.2.1 Percentage of nodes without a viable parent

The following graph illustrates, in a typical simulation run, the number of nodes that are actively seeking a parent (i.e. currently without a stable parent). The state of the nodes are examined at thirty second intervals.

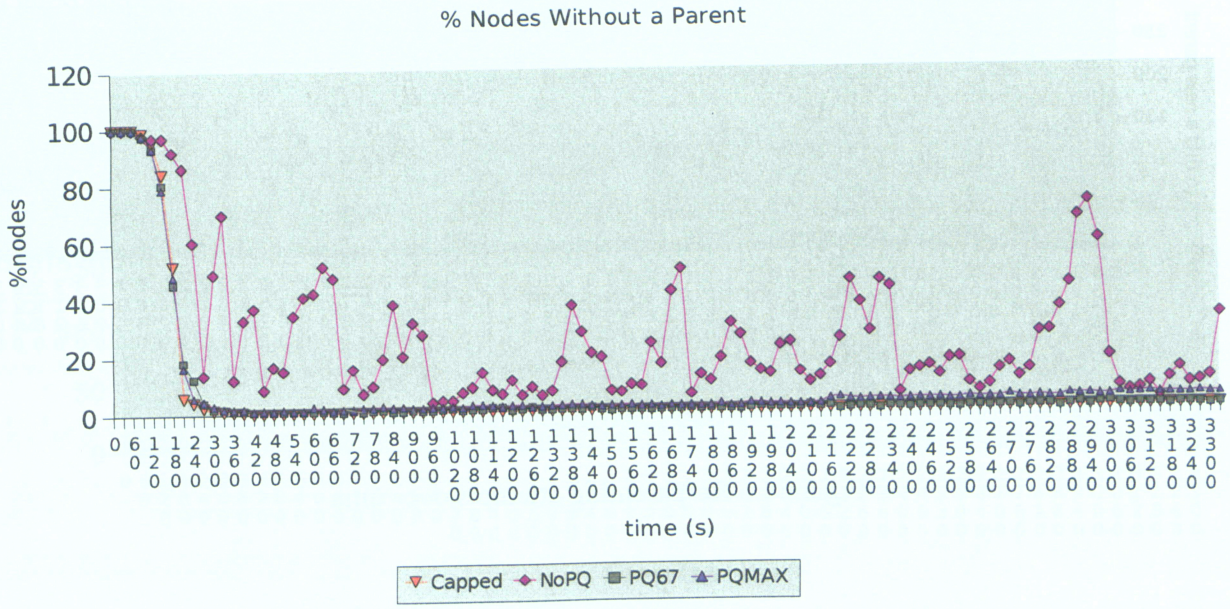


Figure 4.5 Percentage nodes without parents

The superiority of the *pqmax* and *pq67* strategies over *nopq* queue (round robin scheduling) is clearly

evident. When nodes are allowed to peer with other nodes in an unrestricted manner and no priority queue exists, there are large and sudden changes in topology. *Pq67* shows slight improvement over *pqmax* in its ability to mimic the capped performance.

4.2.2 Percentage of nodes without a viable parent

Here we examine the stability of the topology by looking at the number of times where a child who already has a child voluntarily leaves to seek out a "better" parent. The effects of nodes that attempted to join the video stream but were never able to find a suitable parent was thus eliminated.

From the graph below we can once again see the advantages of *pqmax* and *pq67* over simple round-robin scheduling. As compared to the capped strategy, *pq67* once again most closely mimics it with the topology being essentially stable at 300 seconds.

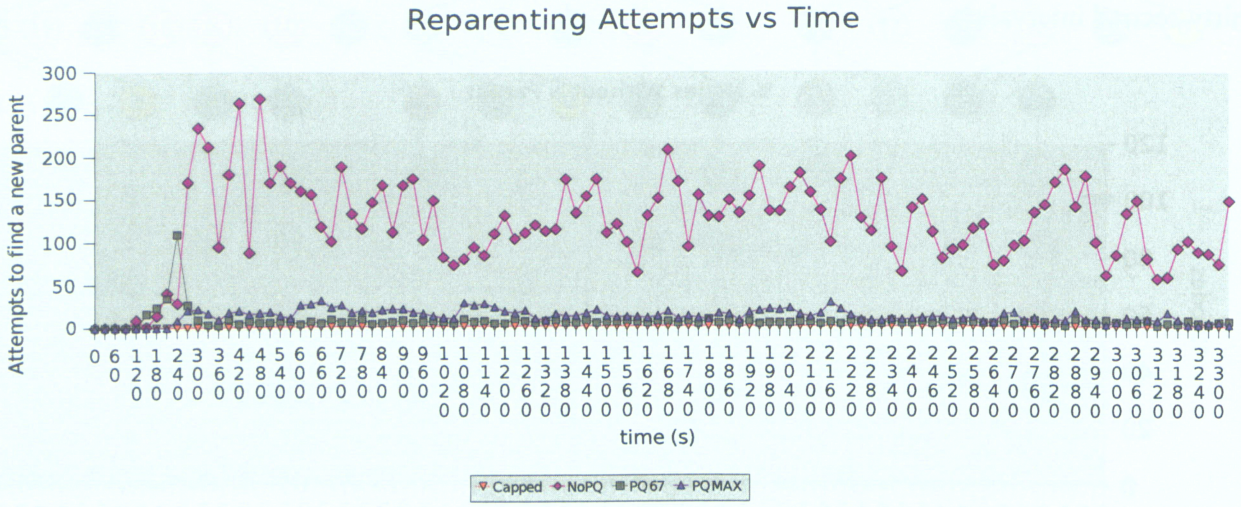


Figure 4.6 Re-parenting attempts over time

4.2.3 Received video bandwidth per node

Possibly the most important of all, the following diagram illustrates DistVid's ability to deliver video. Recall, the target of this topology is to deliver a video stream at 200kbps. For this run of the simulation, pq67 performs the best of all four strategies with the *nopq* strategy being once again clearly inferior. Bandwidth here is a measure of the number of GOPs that a node recovers per second during playback. As mentioned before, each GOPs for the simulations presented here are one second in duration and thus 200kilobits in size.

Note that while it may seem odd that any of the strategies exceed 200kbps it is, in reality, easily explained. A parent will stream a requested segment of video as quickly as the child can receive it. Should the parent not be saturated with children it can and will stream video out to its children above 200kbs. What the graph implies is that the topology resulting from the pq67 strategy has less nodes that have over saturated upload band width (less children) and also that there are fewer nodes unable to find good parents.

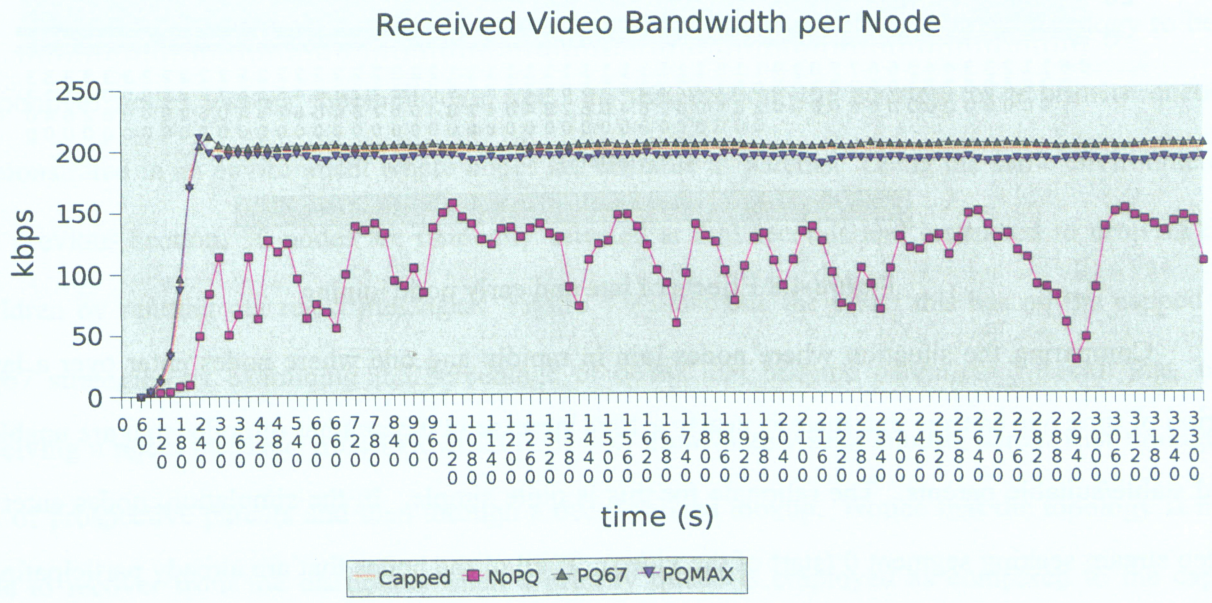


Figure 4.7 Overall received video bandwidth compared

4.2.4 Effects of nodes attempting to join late into the video stream

Employing a priority queue size of 67% we now look at the effects of nodes attempting to join the video stream later on. Prior experiments had made the assumption that, relative to the segment length, nodes attempt to catch the video stream when the show "starts". Thus the majority of nodes attempt to seek parents for the video stream very early on. The chart below looks at two simulation runs. One where 90% of the nodes have entered the video stream by the first 180 seconds compared to a second run where 90% participation is not reached until 720 seconds.

Figure 4.8 plots the percentage of nodes without parents alongside a plot of the percentage of nodes that has joined into the video stream.

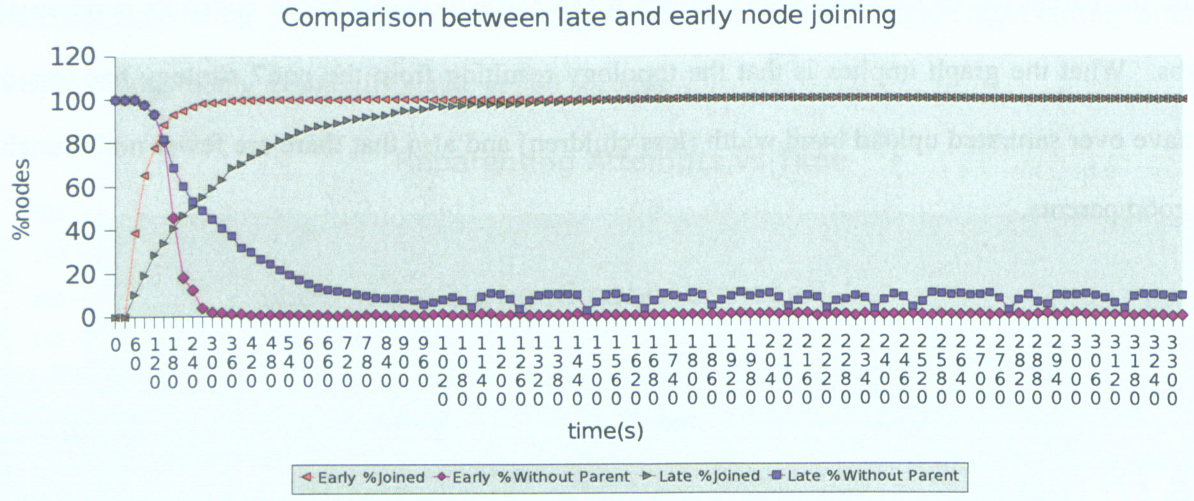


Figure 4.8 Effects of late and early node joining

Comparing the situation where nodes join in rapidly and one where nodes enter over a larger period of time one can see that there is nearly a 10% increase in the number of nodes that are unable to find stable/suitable parents. The rationale for this is quite simple. In the simulation, nodes enter the video stream seeking segment 0 (start of the video). If all of the nodes that are already participating in the video stream have moved to viewing segments 2 or greater. Thus the only source of segment 0 would be the source node who by now has likely been saturated with clients. Thus any new nodes

trying to parent with the source node will not achieve a stable relationship.

Clearly, strategies to minimize this effect need to be implemented. These may include having nodes already on the video stream randomly cache what they have already received increasing the number of available sources. Alternatively, a strategy for newly arriving nodes to issues lookups for progressively higher segment values until a stable parent is found. Note also that nodes choose from available parents in a random fashion thus allowing the source node to become saturated. DistVid can easily be modified such that a node that has the entire contents of a video, in this case our source node, can discourage children node from using it as a parent when seeking out parts that other nodes already have.

4.3 Capped vs PQ67 in an Unstable Node Environment

As mentioned in section 3.5, establishing a priority queue allows a DistVid topology to better respond to nodes leaving the multi-cast tree. In this section, the strength of a priority queue is demonstrated in an environment where nodes are unstable as parents. Using the same environment as the previous section, 50 nodes are randomly selected at 500 seconds and instructed to drop all their children by sending out reject messages. Figure 4.9 illustrates the effect this has on the capped and PQ67 strategies by examining the percentage of nodes that become parentless. Recall that, upon receiving a reject message, a DistVid child node will immediately seek another parent first through its list of prospective parents and then through a decentralized lookup. Notice that the topology is more able to recover from the disruption when a priority queue is employed as compared to the capped strategy.

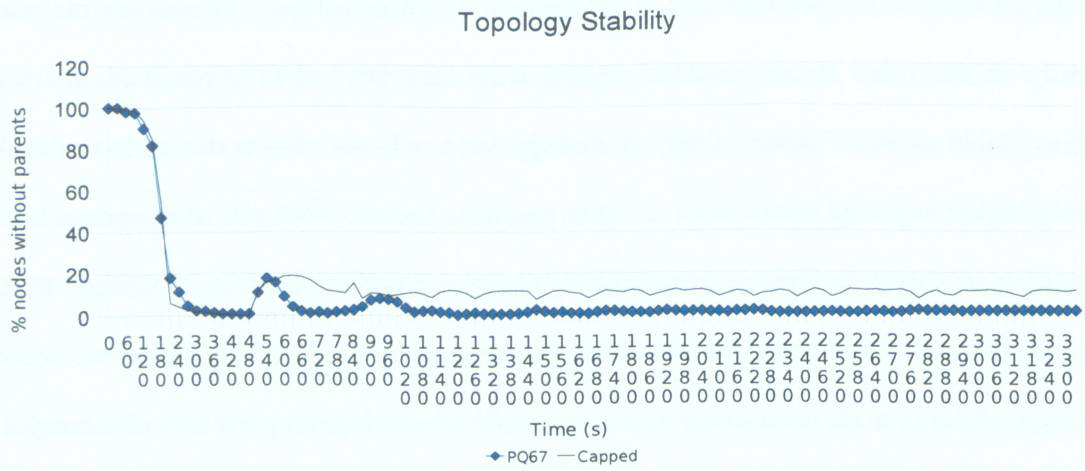


Figure 4.9 Topology Stability After Node Exit

4.4 Effects of Segment Size

From the operating description of DistVid, one can probably deduce that the size of a DistVid video segment can have a dramatic impact on the overall performance of the overlay network. Segment size and thus the number of segments directly impacts factors such as how frequently keep-alive messages are sent, the frequency in which nodes issue DECLARE messages, and the utilization of the Chord overlay.

A very large segment size will concentrate lookups to a small subset of the Chord ring simply because there would be fewer segments. This can make DistVid less capable of recovering from a single node failure quickly. At the same time large segments reduce the amount of overhead network traffic simply because nodes will issue DECLARES less frequently. In contrast, smaller segments distribute the lookup burden over a larger part of the Chord ring. However, as segments get smaller the

burden of overhead network traffic will increase.

The following sections briefly looks at the effects of segment size on a DistVid node's ability to find a stable parent. The simulation was set at 600 Chord nodes with 500 nodes participating in DistVid employing the *pq67* queuing strategy. The participating nodes all enter at the same time and immediately begin looking for a suitable parent to join the video stream starting from time zero. Notation used in the following diagrams are be "number of segments"/"size of segment" or alternatively "number of segments"/"GOPs per segment".

4.4.1 Increasing Segment Size

Figure 4.9 illustrates a direct consequence of increasing the segment size. As the size progress from 30, 90, 120 and finally 360 GOPs with the total number of segments decreasing accordingly, we see an delay in the ability of DistVid nodes in finding a suitable parent quickly.

The primary cause for this is the fact that the size of the individual working buffers used by DistVid are the same size as a video segment segment. A DistVid node will wait until it has a complete segment before it will issue a DECLARE. Thus, despite potentially having a large portion of the video stream, other DistVid nodes, will remain unaware of what that node has and consequently will be unable to locate that node to parent with it. This is especially apparent in the 10/360 plot where the step like plateaus are all roughly the duration of one segment (360 seconds).

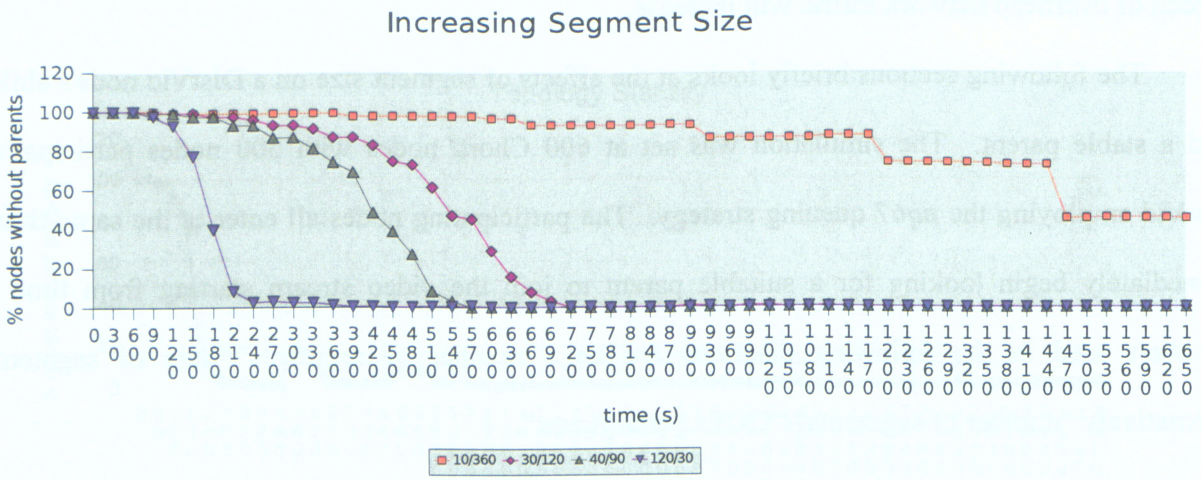


Figure 4.10 Effects of increasing segment size

4.4.2 Decreasing Segment Size

The effects of a small segment size is difficult to illustrate in our simulations because PlanetSim does not model an actual network at the data-link layer. Thus effects, such as, link latencies and packet loss are not present in PlanetSim where messages traveling between nodes instantaneously and always correctly. This is especially important considering that Chord routes between DistVid nodes with $O(\log_2 n)$ overlay hops. Thus, messages for a 600 node overlay can potentially travel to 9 other nodes before reaching its intended destination. In between these overlay hops could potentially be dozens of routers.

When segment sizes are too small, by the time a querying node receives a response to a LOOKUP, the discovered potential parents may have already moved onto different segments. Figure 4.10 illustrates the effects of setting the segment sizes to 30, 15, 10 and 5 GOPs.

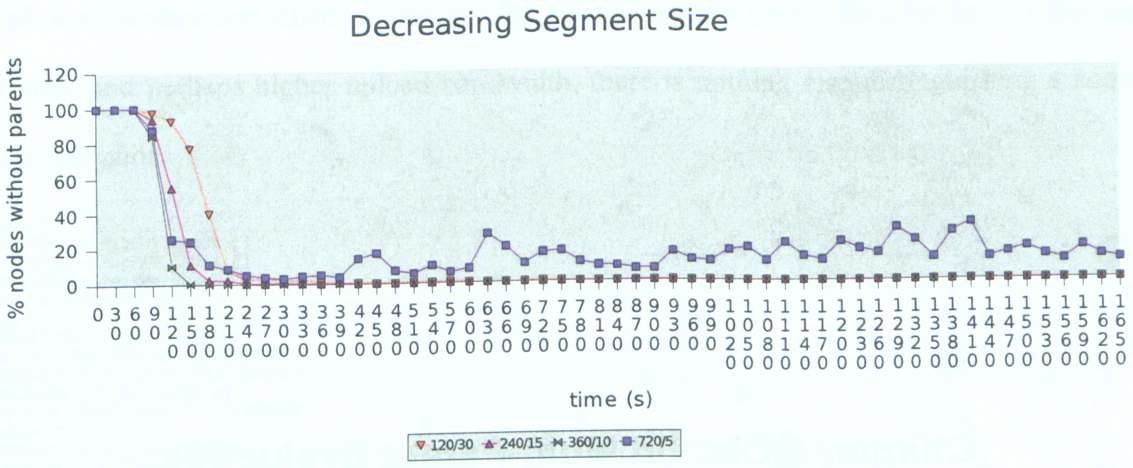


Figure 4.11 Effects of decreasing segment size

As expected, the results are opposite to that of the increasing segment sizes. As the segment sizes decrease, querying nodes are able to find parents more rapidly. However, at 5 GOPs performance of the overlay quickly deteriorates. Nodes have a difficult time locating new parents because each node stable on the video stream are moving through segments too quickly for LOOKUPs to be effective.

Chapter 5 DistVid Multi-Parent Evaluation

The following sections in this chapter, DistVid is extended for multi-parenting operations. To ease comparison, simulations were run keeping the same profile that was used in the single parent scenarios discussed in Chapter 4. The following changes were made to incorporate multiple parenting.

- Video: MDC encoded into 4 distinct descriptions each at 200kbps and 100kbps
- 4 source nodes are present each with the entire contents of one description
- Node leave threshold: 40%
- Good parent threshold: disabled
- Queuing strategy: pq67 priority queue

5.1 Video Stream Topology

Figure 5.1 illustrates the typical peer-to-peer relationships that form under multi-parenting. There are twenty nodes in total, sixteen of which are clients numbered four through fifteen. Nodes zero to three, shown in black, are source nodes with each one hosting exactly one description of the video stream.

Separation of source and client nodes are for clarity reasons only. Besides having the the entire description, and perhaps higher upload bandwidth, there is nothing else distinguishing a source node from a client node.

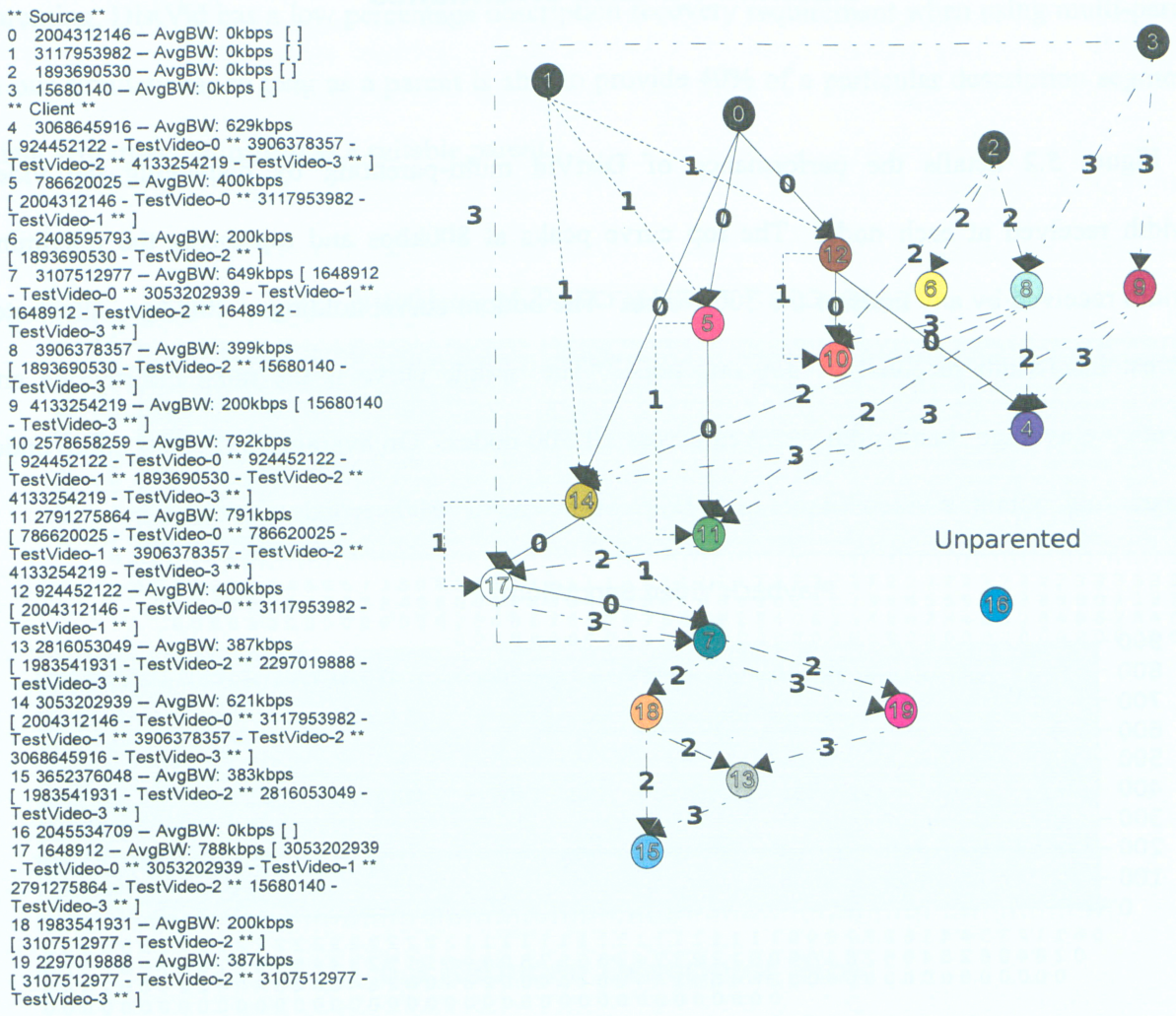


Figure 5.1 Multiple parent topology

5.2 Multi-Parent Performance

Figure 5.2 details the performance of DistVid multi-parenting by displaying the video bandwidth received at each node. The top curve peaks at 800kbps and represents the maximum bandwidth received by any node of the 500 clients. The bottom curve, holding at 200kbps, illustrates the lowest bandwidth received by any one node. The middle curve is the most informative and represents the average bandwidth received across all 500 nodes. On average, a DistVid node, under multi-parenting, obtains a video bit rate of 491kbps at the end of 3300 seconds.

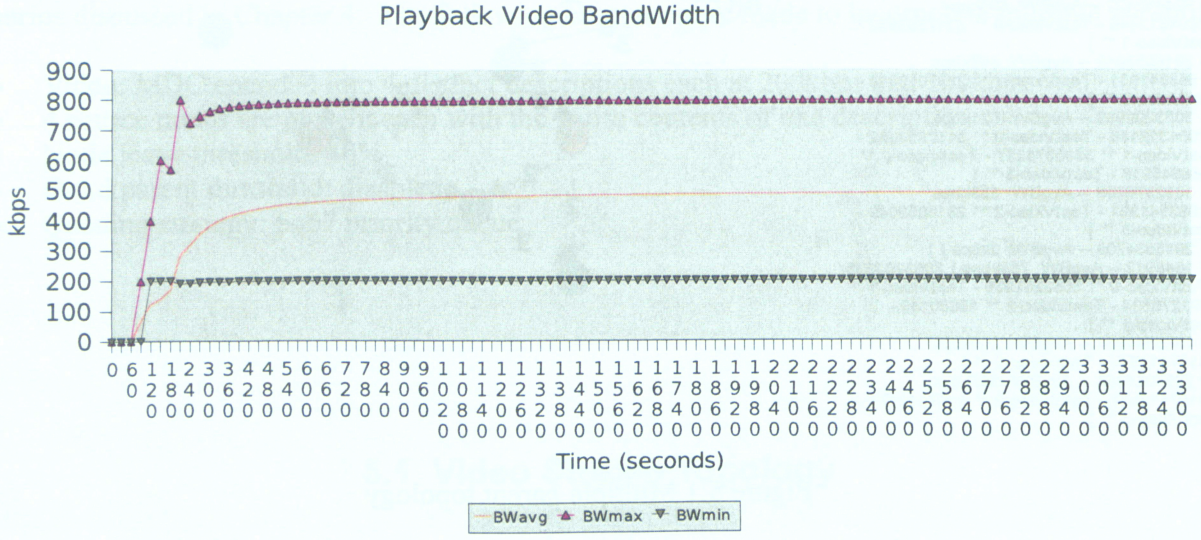


Figure 5.2 Multi-parent playback bandwidth

The above results can be explained by looking at the number of descriptions actually received by a particular node. A node that manages to recover all four descriptions will be expected to enjoy a hearty video bandwidth close to 800kbps. Similarly, a node that is only able to recover one description

will logically receive at most 200kbps. Figure 5.3 illustrates that, in general, a node is able to recover two out of the four descriptions. It should be noted that, while a node may be recovering two descriptions, they may not necessarily be *complete* descriptions. Recall from Chapter 3, unlike single parenting, DistVid has a low percentage description recovery requirement when using multi-parenting. In our simulations, so long as a parent is able to provide 40% of a particular description segment, the client will treat that parent as a suitable parent.

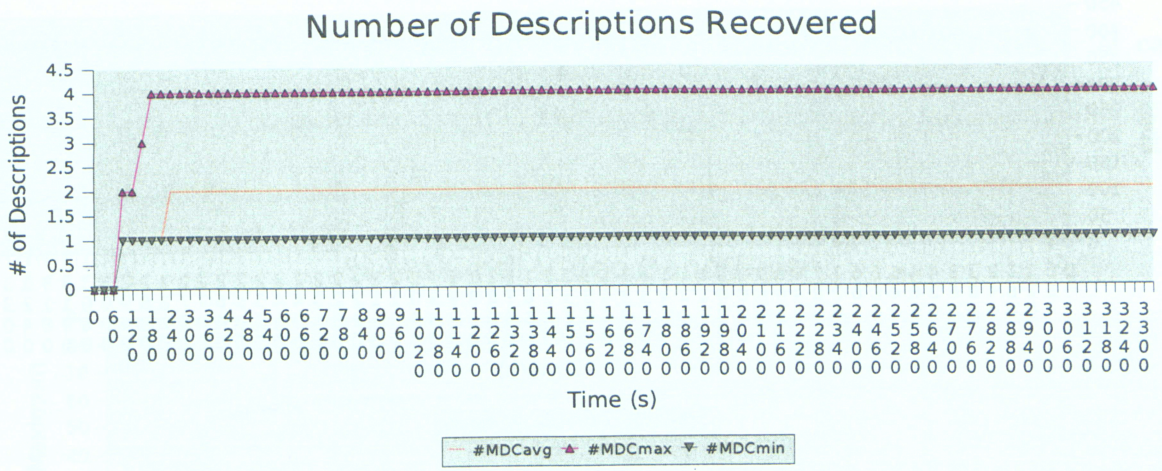


Figure 5.3 Multi-parent descriptor recovery

5.3 Reduced Descriptor Size

From the previous section, the average recovery of only two descriptions out of four may seem quite low. However, for our configuration, it is not surprising. The cause of this is the upload bandwidth of each client node. Set at 600kbps, by employing a pq67 queuing strategy and 200kbps GOPs, we would expect each client node to reliably and completely serve two children. Any extra

children must then compete for residual bandwidth.

By decreasing the descriptor size, the ability of a node to serve more children should increase given the reduction in bandwidth demand per child. Figure 5.4 and 5.5 illustrates the effects of keeping the same setup but reducing the descriptor size to 100kbps. Note the improvement in the ability of nodes to recover video descriptors as nodes are on average able to recover three out of the four descriptors. Video bandwidth increases correspondingly relative to the maximum bandwidth..

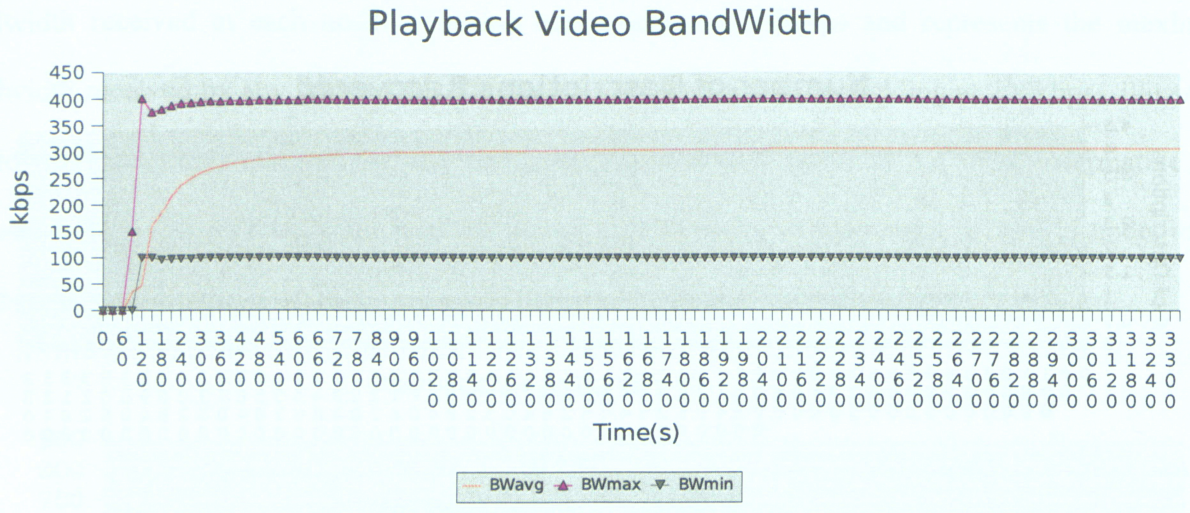


Figure 5.4 Video bandwidth for 100kbps descriptor

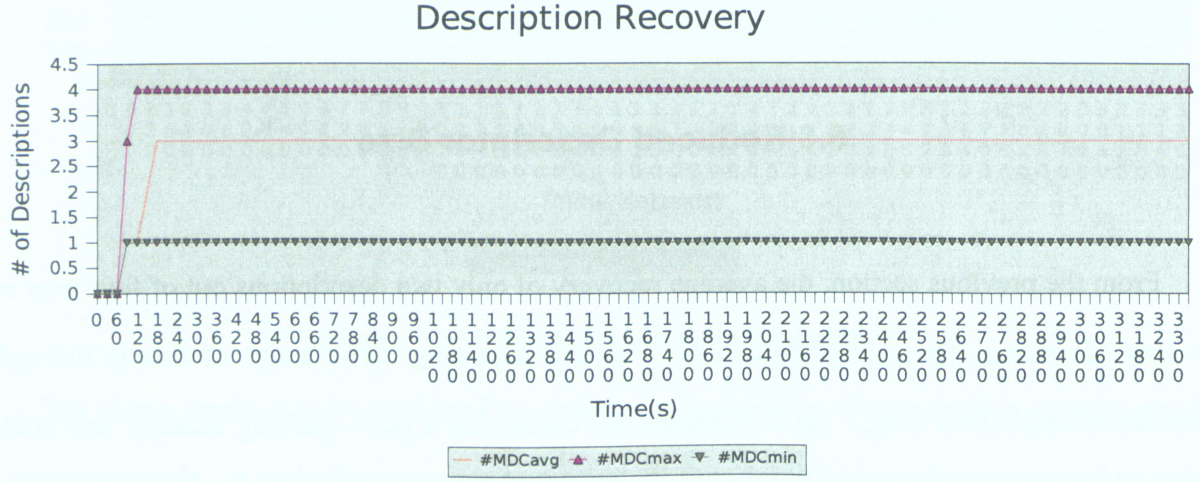


Figure 5.5 Number of descriptors recovered for 100kbps descriptors

Observe that the recovered video bandwidth closely mimics the number of descriptors recovered. The bandwidth is however slightly lower than the theoretical maximum bandwidth for a given number of descriptors. As mentioned before, the node may not be able to recover the full descriptor set for a given segment but is instead combining multiple incomplete descriptors to maximize image quality.

Figure 5.6 compares the the ability of the system to achieve its maximum theoretical video bandwidth. For the 100kbps descriptor, this would be 400kbps and the 200kbps descriptor, 800kbps. The lower descriptor GOP sizes allows the system to much more effectively deliver at capacity achieving 77% capacity as compared to 61%. This once again illustrates the increased ability of nodes to service more children by having a descriptor that imposes a smaller bandwidth burden

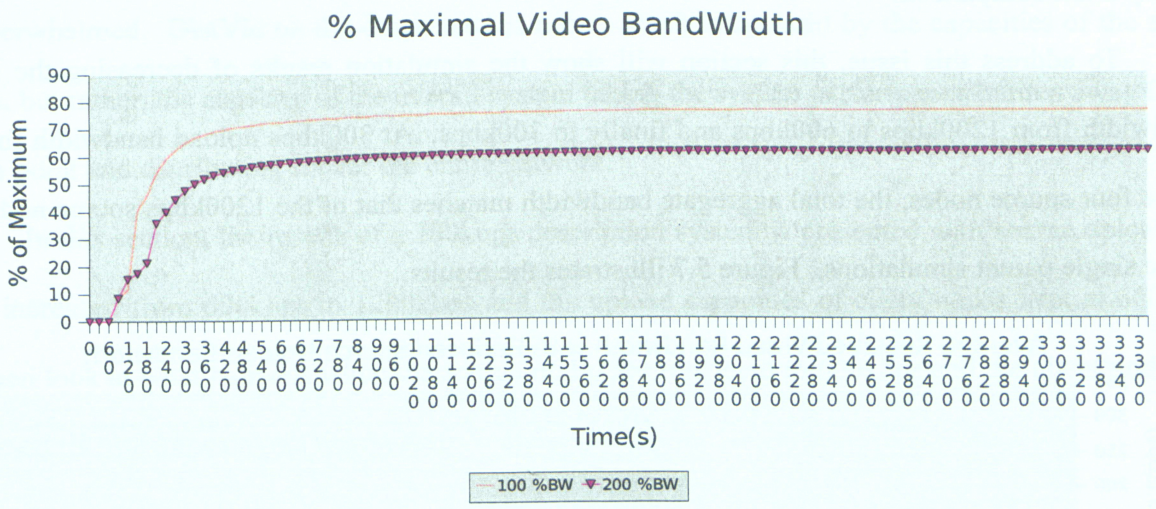


Figure 5.6 Percentage of maximum bandwidth achieved

5.4 Multi-Parenting Compared to Single-Parent Model

Sections 5.2 and 5.3 implies that multi-parenting is more effective than the single parent model, being able to deliver an average video of data rate of 491kbps and 306kbps for the 200kbps and 100kbps descriptors respectively. This is in contrast to the 200kbps *maximal* data rate for the single-parent configuration. However, one may argue that these are very different environments in which DistVid is being compared especially since there are now four video sources each with a 1200kbps upload capacity. The aggregate upload bandwidth of the source notes is now peaks at 4800kbps. Hardly a fair comparison.

To address this issue, this section will show the simulation results of decreasing the source bandwidth from 1200kbps to 600kbps and finally to 300kbps. At 300kbps upload bandwidth for each of the four source nodes, the total aggregate bandwidth matches that of the 1200kbps source node used in the single parent simulations. Figure 5.7 illustrates the results.

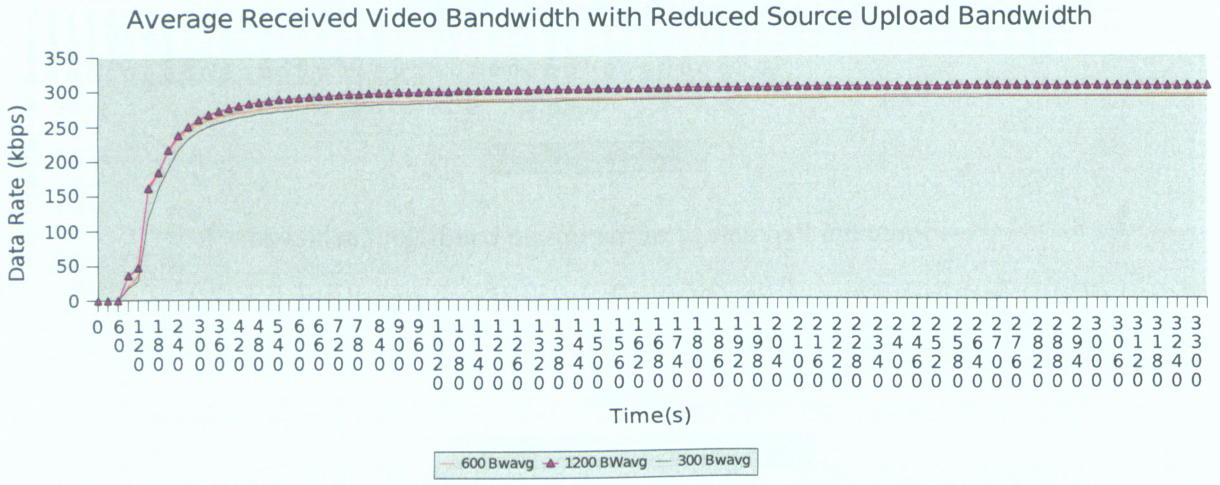


Figure 5.7 Effects of reduced source upload bandwidths

Notice that the impact of reducing the source upload bandwidth is not large. Even when reduced to only 300kbps, the DistVid multi-parent model is able to deliver an average data rate of 290kbps, a 45% increase in video data rate delivery compared to the single-parent model.

5.5 Effects of bandwidth at the Source Node

Section 5.4 illustrates one the original design goals of DistVid. In the traditional client-server model, the performance of the system is directly tied to the ability of the server to deliver content. Namely, the server's upload capacity. Thus as bandwidth decreases or demand increases, the server can be overwhelmed. DistVid on the other hand is not so much influenced by the capacities of the source nodes, but rather, the capacity of the overall system taking the system performance burden away from a single point and distributing it over the entire network.

In this section, the results of a 200kbps description system is presented with source upload data rates increased from 600kbps to 1200kbps and the upload capacities of client nodes kept at 600kbps. We then look at what happens when the source nodes are restricted to 600kbps and the client nodes are increased to 1200kbps upload bandwidth.

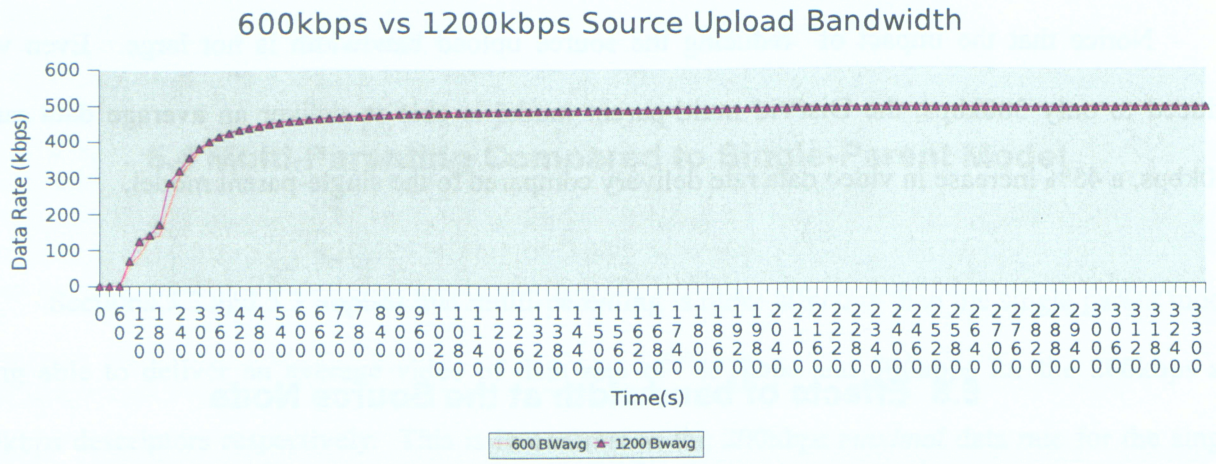


Figure 5.8 Source upload bandwidths compared

Observe from Figure 5.8 that there is little to no impact on overall system performance by doubling the source upload bandwidth. In figure 5.9 we overlay the graph where we keep the source nodes at 600kbps upload bandwidth and increase the client nodes to have 1200kbps upload capacities. The improvement is system performance is easily visible as average received video bandwidth improves by nearly 20%.

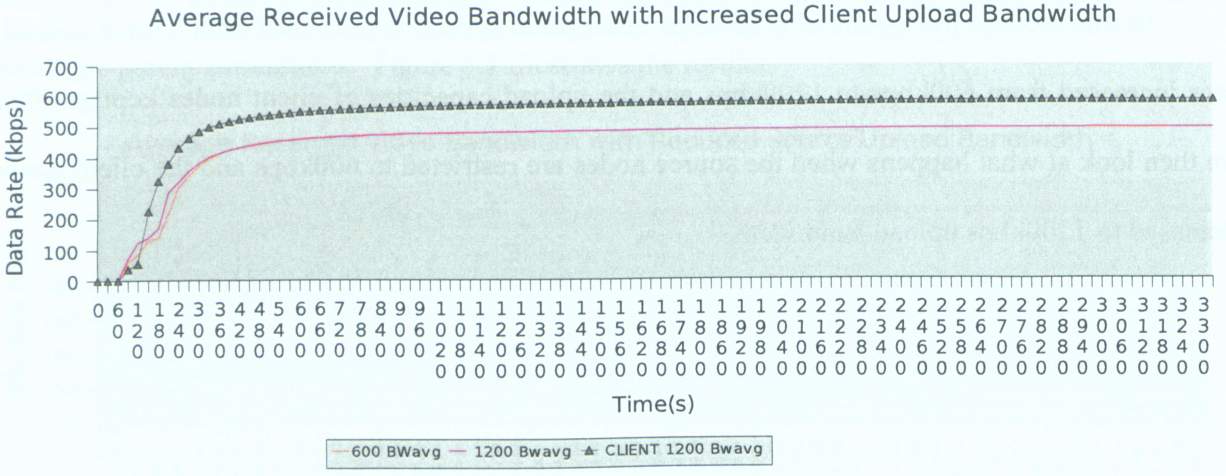


Figure 5.9 Effects of increased client bandwidth

Client bandwidths is clearly a much more influential factor in overall system performance than the source bandwidth in a system such as DistVid.

5.6 Node Distribution by Video Bandwidth

In the final section of this chapter, we examine the distribution of nodes based on the video bandwidth they receive. Using the notation of "source bandwidth"/"client bandwidth" we examine the distribution of nodes based on the percentage of the theoretical maximum bandwidth at intervals of 10%. Topologies using 600kbps/600kbps, 300kbps/600kbps, and 1200kbps/600kbps at 100kbps MDC encoding, and 600kbps/200kbps at 200kbps encoding.

As can be seen from the following diagram, affirming what was discussed in section 5.5, the performance of the overall system is similar for each run despite the change in the source upload bandwidth. In general, from multiple simulations run, over 50% of participating nodes were able to achieve greater than 70% of the theoretical maximum bandwidth.

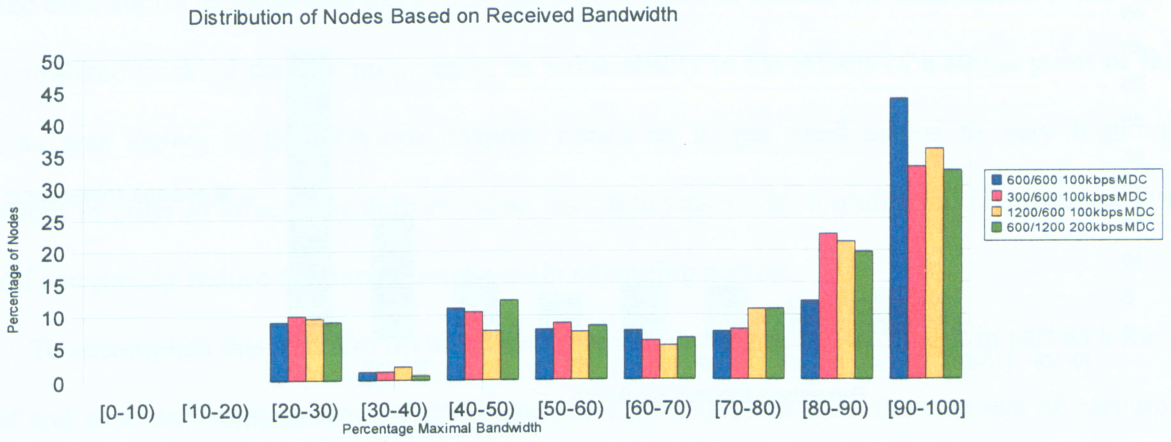


Figure 5.10 Distribution of nodes by received bandwidths

Note that the above figure displays the results of nodes that were able to locate at least one parent for at least one description of the video. For these nodes, not a single node recovered less than 20% of the video bandwidth. This result should be taken with a caveat. A DistVid node is set to abandon a source that is unable to deliver at least 40% of the segment in the duration of one segment.

If such nodes are only able to retrieve one description they would attempt to seek another parent in the process becoming parentless and thus excluded from these results.

Another irregularity that should be pointed out is the low percentage of nodes in the 30-40% recovery category relative the the higher percentage in the 20-30% range. This is because DistVid nodes are set to leave its parent whenever the parent is unable to deliver at least 40% of a segment in the duration of one segment. The low values at the 30-40% range are indicative of this. At the same time, the 20-30% and 40-50% have a higher proportion of nodes because that range encompasses the region where nodes are able to recover fairly well a single description of the video (25%) and two descriptors (50%) respectively.

The diagram that follows illustrates the effects when the simulator is set to have nodes abandon its parent at the lower threshold of 20% as compared to 40%.

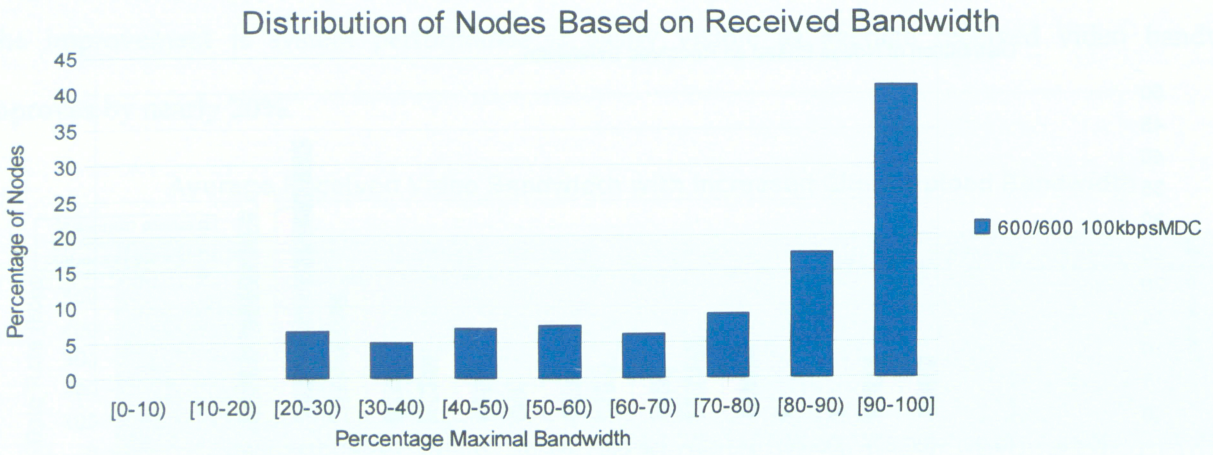


Figure 5.11 Distribution of nodes by received bandwidths using lower threshold

Chapter 6 Conclusion

DistVid is a proposed streaming protocol that provides a decentralized alternative to traditional server client models by combining the file-sharing abilities of distributed p2p file sharing networks, with the efficiencies of multi-cast trees. The benefits of DistVid include the elimination of the need for any centralized tracking thereby minimizing its vulnerability to the affects of a single point of failure. Also, as was shown in the previous chapter, nodes no longer need access to very high upload bandwidth in order to effectively deliver video. In conjunction with a traditional client-server system, DistVid can greatly reduce the burden on the main streaming servers.

To accomplish this, DistVid divides video into smaller segments, treating it in part as a file to be shared and as a multi-cast stream. DistVid uses the Chord DHT to track the contents of participating nodes and to allow individual nodes to locate suitable places in the multi-cast tree to retrieve the content that it is seeking. The work of this thesis, to develop, and test the performance of DistVid on PlanetSim, provided positive results. Despite not having centralized tracking, stable trees are formed in the presence of new nodes entering and moving within the network. With multiple description coding and multi-parenting, DistVid allows for a much higher and efficient utilization of network bandwidth to

deliver video in a manner that is no longer entirely dependent on the upload capacity on any particular server.

To find a use for such a system, one need not look far. Imagine an existing video streaming infrastructure. To scale such a system for more users may be prohibitively expensive in both server costs and network bandwidth costs. Similarly a small start up company or a community run system, attempting to provide a YouTube like service, may not have the capital or the ability to support the traditional client-server approach. DistVid can be a viable alternative that will scale well as demands grow.

However, much development and testing still remains to be done on a system such as DistVid. Questions that remain to be answered in future work include DistVid's resilience to random node failures, the proper balance between network overhead and segment size, the balance between video quality and the number of MDC streams, and the benefits of forwarding latency information from the DHT layer to DistVid. Ultimately there will be a need to test DistVid in an environment that closely mimics the "real world" environment such as, PlanetLab. Fortunately, with PlanetSim's API and tier structure, when the time comes, the migration of DistVid to PlanetLab should be relatively straight forward.

Glossary

Bittorrent	A peer-to-peer file sharing protocol
CAST	Group based any-cast/multi-cast – a structured network overlay
Chord	A scalable peer-to-peer lookup protocol implementing a DHT
DHT	Distributed Hash Table – a structured network overlay
DistVid	Distributed Video protocol
DOLR	Distributed Object Location and Routing – a structured network overlay
Freenet	A decentralized peer-to-peer network using key based routing
Gnutella	A popular file sharing protocol
GOP	Group Of Pictures
MDC	Multiple Description Coding
Pastry	A distributed object location and routing protocol for wide-are peer-to-peer applictations
Scribe	A decentralized protocol implementing a large-scale application-level multi-cast
SHA-1	One of five cryptographic hash functions designed by the National Security Agency (NSA)

References

[1] X. Xu, Y. Wang, S. Panwar, and K. Ross, "A Peer-To-Peer Video-On-Demand System Using Multiple Description Coding and Server Diversity", *International Conference on Image Processing*, vol. 3, October 2004, p.1759-1762

[2] J. McQuillan, and I. Richer, "The New Routing Algorithm for the ARPANET", *IEEE Transactions on Communications*, vol.28, no.5, May 1980, p.711-719

[3] B. Leiner, and V. Cerf, "A Brief History of the Internet", *IEEE Annals of the History of Computing*, vol. 22, no.2, June 2000, p.77-79

[4] P. Kirstein, "Early experiences with the Arpanet and Internet in the United Kingdom", *IEEE Annals of the History of Computing*, vol.21, no.1, March 1999, p.38-44

[5] S. Androutesellis, and D. Spinellis, "A survey of peer-to-peer content distribution technologies", *ACM Computing Surveys*, vol.36, no.4, 2004, p.335-371

[6] R. Schollmeir, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", *Proceedings of the First International Conference on Peer-to-Peer Computing*, August 2001, p.101-102

[7] S. Merugu, S. Srinivasan, E. Zegura, "Adding structure to unstructured peer-to-peer networks: the use of small world graphs", *Journal of Parallel Distributed Computing*, vol.65, no.2, 2005, p.142-153

[8] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network", *Proceedings of International Conference on Peer-to-Peer Computing*, vol.101, 2001, p.99-100

[9] R. Schollmeir, and G. Schollmeir, "Why Peer-to-Peer (P2P) does scale: an analysis of P2P traffic patterns", P*roceedings Second International Conference on Peer-to-Peer Computing*, 2002, p.112-119

[10]I. Stoica, R. Morris, "Chord: A Scalable Peer-to-Peer Lookup Service For Internet Applications", *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol.31, no.4, August 2001, p. 149-160

[11]S. Ratnasamy, and P. Francis, "A Scalable Content Addressable Network", *Proceedings of the*

2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, vol.31, no.4, August 2001, p.161-172

[12]A. Rowston, and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing For Large-Scale Peer-to-Peer Systems", *ACM International Conference on Distributed Systems Platforms*, vol.11, 2001, p.329-350

[13]M. Castro, and M. Costa, A. Rowstron, "Performance and dependability of structured peer-to-peer overlays", International Conference on Dependable Systems and Networks, 2004, p.9-18

[14]M. Janic, and P. Mieghem, "On Properties Of Multi-cast Routing Trees", *International Journal of Communication Systems*, vol.19, November 2005, p.95-114

[15]T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms 2nd ed*, McGraw-Hill, p.222

[16]I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", *Lecture Notes in Computer Science*, vol.2009, 2001, p.46-63

[17]H Balakrishnan, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data In P2P Systems", vol.46, no.2, 2003, p.43-48

[18]B. Haskell, A. Puri, *Digital Video: An introduction to MPEG-2*, Kluwer Academic Publishers, New Jersey, 1996, p.14-28

[19]ISO MPEG-2 Standard, ISO/IEC 13818-1:2000

[20]V. Goyal, "Multiple Description Coding: Compression Meets the Network", *IEEE Signal Processing Magazine*, vol.18, no.5, p.74-93, September 2001

[21]B. Jenkins, "A New Hash Function for Hash Table Lookup", *Dr. Dobb's Journal*, 1997

[22]S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, "A Survey of Peer-to-Peer Network Simulators", *Proceedings of the Seventh Annual Postgraduate Symposium*, Liverpool, UK, 2006

[23]P. Garcia, C. Pairot, R. Mondejar, and J. Pujol, "PlanetSim: A New Overlay Network Simulation Framework", *Lecture Notes in Computer Science*, vol.3437, 2005, p123-136

[24]D. Stutzbach, R. Rejaie, "Understanding Churn in Peer-to-Peer Networks", *Proceedings of the 6th ACM SIGCOMM on Internet measurement*, 2006, p.189-202

[25]A. Gupta, B. Liskov, and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays", *Proceedings of the 9th conference on Hot Topics in Operating Systems*, vol. 9, p.2-2

[26]Chord FAQ, http://pdos.csail.mit.edu/chord/faq.html

[27]P. Pietzuch, J. Ledlie, M. Mitzenmacher, and M. Seltzer, "Network-Aware Overlays with Network Coordinates", 26th *IEEE International Conference on Distributed Computing Systems Workshops*, July 2006, p.12-12

[28]D. Karger, E. Lehman, and T. Leighton, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web", *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, p.654-663

[29]V. Rijmen, E. Oswalkd, Update on SHA-1, *Lecture Notes in Computer Science*, vol.3376, 2005

[30]X. Wang, A. Yao, and F. Yao, "New Collision search for SHA-1", *CRYPTO Rump Session*, vol. 5, 2005

[31]Karger, D., Lehman, E., "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web", *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, p.654-663

[32]S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi, "A Statistical Theory of Chord under Churn", *The 4th Annual International Workshop on Peer-to-Peer Systems*, February 2005

[33]S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz, "Handling Churn in a DHT", *Proceedings of the USENIX Annual Technical Conference*, June 2004, p.127-140

[34]Bittorrent Protocol, http://www.bittorrent.org, 2007
[35]W. Poon, J. Lee, and D. Chiu, "Comparison of Data Replication Strategies for Peer-to-Peer Video Streaming", *Fifth International Conference on Information Communications and Signal Processing*, December 2005, p.518-522
[36]Hales, D., Patarin, S., "How to cheat bittorrent and why nobody does", *Technical Report UBLCS-2005-12*, 2005, p.1-8
[37]F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System", *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol.34, no.4, August 2004, p.15-26
[38]F. Fitzek, B. Can, and R. Prasad, "Overhead and Quality Measurements for Multiple Description Coding for Video Services", *Wireless Personal Multimedia Communications*, Sept, 2004
[39]YouTube, http://www.youtube.com, 2007
[40]W. Li, "Overview of fine granularity scalability in MPEG-4 video standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.11, no.3, March 2001, p301-317
[41]H. Zhang, A. Goel, and R. Govindan, "Improving Lookup Latency in Distributed Hash Table Systems Using Random Sampling", *IEEE/ACM Transactions on Networking*, vol.13, no.5, October 2005, p.1121-1134
[42]Revver, http://www.revver.com, 2007