OmniStream: Using Centralized Tree Management for Incentives-Compatible Peer-to-Peer Media Streaming

by

Ankur Upadhyaya

B.Sc., University of British Columbia, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

October 2008

Abstract

The ability to exploit Internet infrastructure to support the live delivery of streaming media to large audiences would be of significant practical value. Existing approaches to this problem, however, based on the use of IP Multicast, commercial content distribution networks, or costly bandwidth provisioning, are severely limited: IP Multicast has seen only sparse deployment, while the latter two options are prohibitively expensive for many content providers. In recent work, peer-to-peer media streaming has been explored as an interesting alternative solution. Here, a server peer forwards a stream to a number of client peers using basic unicast transmission. Clients receiving the stream can then contribute their unused upstream bandwidth to forward the stream to additional clients, who in turn can do the same and so on. In effect, the bandwidth demands of a streaming session are distributed over the set of receivers; although each of these collaborative end-hosts may have a low-capacity access link, together they essentially "pool" their upstream bandwidth to support large-scale streaming sessions.

Unfortunately, this cooperative approach to media streaming is vulnerable to the inherent tension between selfish interests and collective welfare characteristic of peer-to-peer systems. The potential prevalence of free-riding, whereby a peer consumes bandwidth without adequate reciprocity, is of particular concern. Given the highly sensitive throughput requirements of media streaming, the scalability of any peer-to-peer approach would be severely limited by such uncooperative behavior. Although a handful of mechanisms have been proposed to address this issue, they are limited either by their reliance on "tamper-proof" software - whereby they unrealistically assume that peers will correctly follow a prescribed protocol even though it may be in their selfish interest to deviate, or other practical considerations (e.g. slow responsiveness, instability). In this thesis, OmniStream, a solution based on the centralized management of multicast trees, is proposed to address the issue of free-riding in peer-to-peer media streaming, without these limitations. The proposed protocol is evaluated through simulation experiments representative of typical operational scenarios. The results obtained suggest the viability of the proposed approach.

Contents

List of Tables

List of Figures

Acknowledgements

I wish to express my sincerest gratitude to my supervisor and mentor Dr. Son Vuong. His constant encouragement, patience, guidance and wisdom throughout the ups and downs of my research was invaluable. Most of all, I thank him for the considerable trust and latitude he has given me over the course of my work; in allowing me the freedom to explore and occasionally stumble down the many avenues I encountered along the way, I have learned a great deal under his supervision.

I would also like to thank my friends in the Network and Internet Computing Lab and the Distributed Systems Group. Their insightful perspectives and stimulating influence has done much to further my research. I especially appreciate the contributions of my second reader Dr. Charles Krasic; this thesis has benefited tremendously as a result of his thoughtful feedback.

In addition, I gratefully acknowledge the generous financial support I have received from the National Sciences and Engineering and Research Council of Canada, the British Columbia Advanced Systems Institute and the Faculty of Graduate Studies.

Finally, as always, I am deeply thankful for the unwavering love, care and support my family and friends have blessed me with throughout. Without them, this thesis would never have been possible.

ANKUR UPADHYAYA

The University of British Columbia
October 2008

To the memory of my loving grandmothers.

CHAPTER 1

Introduction

As observed in [7], the vision of using the global Internet infrastructure to support live media broadcasts from arbitrary content providers to arbitrary sets of receivers has been an emphasis of networking research for well over a decade. (We note that the term "live", as used in this context, refers to the simultaneous distribution of the same stream to multiple clients – the content itself may be live or simply the playback of a file.) Such an Internet utility would have numerous advantages over traditional media dissemination schemes (e.g. radio, cable, satellite). For example, by making use of a global network infrastructure, live Internet streaming would not suffer from the limited geographical reach to which radio and television stations are often restricted. Further, by making broadcast a common Internet utility and providing support for arbitrarily large numbers of simultaneous streaming sessions, "channels" would no longer be limited to those provided by a handful of wealthy content publishers. Finally, Internet broadcasts can make more efficient use of infrastructure by only serving interested participants. We note that the realization of this vision has also been given renewed impetus by a number of recent technology trends, including: (1) the continued proliferation of broadband Internet connectivity (e.g. Cable, DSL) – as of March 31, 2005 some 164.4 million broadband lines were in use, this represents an increase of 8.5% over the previous figure of 151.5 million lines just three months prior [29], (2) the rapidly growing popularity of streaming media applications, as well as (3) the imminent ubiquity of multimedia capable networked devices (cellular phones, personal digital assistants, video sensor networks, etc.).

IP Multicast [10], a proposed extension to the Internet architecture, was originally touted as a network-level (i.e. router-level) solution to the problem of supporting live streaming to multiple clients. Unfortunately, in spite of the fact that IP Multicast has been implemented and is available in most routers today, this technology has seen only sparse deployment as a result of a number of fundamental limitations [5]. First, in requiring routers to maintain state on a per-multicast group basis, IP Multicast introduces undue complexity and serious scalability concerns. Second, experience has shown that, unlike the unicast case, it is incredibly difficult to implement higher-level transport layer functionality (e.g. flow, error and congestion control, security) on top of the best-effort semantics provided by IP Multicast. Finally, the requirement for infrastructure level changes has naturally delayed acceptance.

In the absence of IP Multicast, content providers may invest in high-capacity infrastructure, or subscribe to a commercial content distribution network (CDN) - neither option is cost effective. In the case of the former, we note that in the absence of network support for multicast, the content provider's server infrastructure – perhaps a single machine, or a distributed cluster – must unicast a replica of the stream to each individual client. Given the duration and high throughput requirements of a typical media stream - a reasonable quality audio/video stream takes 350kbps to encode [8], this inefficient use of bandwidth is highly problematic: while an insufficiently provisioned access link will not be able to support large audiences, a higher capacity will be prohibitively expensive for many potential content providers and even then, is not immune to being overwhelmed in the event of a sudden surge in demand (i.e. a "flash crowd"). Commercial CDNs (e.g. Akamai Technologies [2], Digital Island [12], Real Networks [32], Speedera Networks [36], etc.) address this issue by maintaining a large, geographically-distributed network of adequately provisioned servers and allowing content providers to subscribe to this infrastructure. Unfortunately, the CDN model is inappropriate for many publishers. For small- and medium-scale publishers, normal traffic levels rarely justify the cost of CDN subscription. Further, as noted in [27], there is even increasing evidence that migration to in-house server-farms may be more economical for large-scale publishers [23]. (Although not further explored in this work, we note that newly emerging approaches to content distribution based on the use of cloud computing infrastructure – e.g. justin.tv [20] – may represent a practical, cost-effective option.)

Peer-to-peer architecture presents an interesting alternative to the approaches above. Under such a design, a server peer acting as a content provider forwards a stream to a number of client peers using basic unicast transmission. Again, because a copy of the stream is unicast for each client, a peer with a low upstream capacity access link will only be able to directly service a relatively small number of client peers. Nevertheless, client peers receiving the stream can in turn use their unused upstream bandwidth to forward the stream to additional clients, who in turn can do the same – with this hierarchical forwarding scheme potentially continuing over any number of levels and encompassing arbitrarily large numbers of participants. In effect, the bandwidth demands of a streaming session are no longer concentrated at the content provider but rather, are distributed over a set of collaborative end-hosts. Although each of these individual end-hosts may have low-capacity access links, they effectively "pool" their upstream bandwidth to support large-scale streaming sessions. In short, a peer-to-peer approach allows bandwidth to scale with demand – live broadcasts to large audiences are possible without prohibitive investments in infrastructure.

The notion of peer-to-peer media streaming has stimulated a flurry of recent research activity. Numerous architectures have been proposed and/or implemented, some examples of which include: End-System Multicast (ESM) [5], CoopNet [27], PROMISE [17], SplitStream [4], ZIGZAG [39], P2P-TV [30], Peer-to-Peer Receiver Driven Overlays (PRO) [33], PeerCast [28], GnuStream [19], PeerStreaming [24], oStream [9], Dagster [25], PPLive [31], as well as numerous other frameworks [3, 11, 42]. Of these, few have seen any appreciable practical deployment - clearly, significant research challenges remain.

At least one major class of problems arises from the fact that while peer-to-peer media streaming depends on peers using their upstream bandwidth to forward data, it is unreasonable to expect that they will obediently behave as routers. Rather, these peers are end-user software processes and as such, may implement a variety of uncooperative behaviors. Although the space of possible such behaviors is large, experience in the peer-to-peer file sharing domain indicates that the phenomenon of "free-riding", whereby a peer consumes resources without contributing an equal of greater amount, is of particular concern. Measurement studies of the Gnutella file-sharing system have shown that fewer than 30% of users contribute any resources at all and that the majority of downloads are serviced by only 1% of hosts [1]. Game-theoretic analysis of such systems has also been used to show that free-riding is to be expected as a naturally emerging effect, as well as to conjecture that the prevalence of such behavior increases with the scale of a system [14]. While file-sharing is possible under such tenuous conditions, peer-to-peer media streaming, with its considerably more sensitive bandwidth requirements, would likely collapse – a fact that has been verified in simulation [15].

Despite extensive exploration of the free-riding problem in the context of file-sharing, only a handful of studies relevant to streaming are available – none of which present a comprehensive solution. Many of the anti-free-riding incentive mechanisms outlined in this previous work (e.g. [6, 8, 14-16]), assume the use of "tamper-proof" proprietary software, which in turn, ensures that peers adhere to the distributed protocol used to realize the mechanism. Clearly, however, this assumption is unreasonable in the case of a widely deployed Internet streaming utility – on such a large scale, the implementation of peers that deviate from the prescribed protocol for selfish gain must be anticipated. Mechanisms that do not rely on the assumption of tamper-proof software are presented in [26] and [35]. While the details of these schemes are deferred to the following chapter, we note that the first forces a tradeoff between high protocol overhead and slow responsiveness to free-riders - and further, breaks down when free-riders exhibit even a limited degree of sophistication (e.g. free-riding only sporadically) – while the second is susceptible to a high degree of instability.

In this thesis, OmniStream - an extension of the CoopNet [27] peer-to-peer media streaming framework - is developed to address the free-riding problem. In the proposed scheme, a single media stream is encoded as a number of sub-streams, also referred to as "descriptions", such that a client receiving some non-empty subset of these sub-streams can at least partially reconstruct the original stream, with higher quality being achieved as more descriptions are used. Each sub-stream is then disseminated along an application-level multicast tree whose topology is directed by the content provider. By centrally managing the overlay, the content provider has complete knowledge of its topology and so, can determine the contribution-level of each client simply by counting how many descriptions it is consuming and how many it is forwarding. With control of the overlay topology, the content provider can also determine the quality of service enjoyed by each peer, by manipulating the number of multicast trees it is granted membership to - and hence, how many descriptions it is able to receive - as well as its distance from the root in each of those trees. By ensuring that peers demonstrating good behavior over time are incorporated into the multicast trees with greater priority, OmniStream is able to neutralize the damaging effects of free-riders. As OmniStream only depends on the content provider's *direct* knowledge and control of the overlay topology to accomplish this, it suffers from neither a requirement for tamper-proof software, nor poor responsiveness.

The remainder of this thesis is organized as follows. Chapter 2 surveys relevant previous research, outlining a number of mechanisms specifically aimed at ensuring fair upstream bandwidth contributions in a peer-to-peer media streaming environment. A detailed specification of the OmniStream framework is provided in Chapter 3. In Chapter 4, the simulation setup used to evaluate this scheme is described and the experimental results obtained are presented and discussed. Finally, Chapter 5 concludes this thesis and outlines some potential avenues of future research.

CHAPTER 2

Related Work

The sub-sections below survey the available research into the design of mechanisms encouraging cooperative behavior in peer-to-peer media streaming applications. Existing schemes are presented in detail, as an understanding of certain salient features is required to identify their limitations and later develop the mechanism proposed in this work.

2.1 Rank-Based Peer Selection

In [14-16], a mechanism that encourages cooperative behavior by ensuring that peers with greater levels of contribution enjoy higher-quality downloads, is introduced. Under this scheme, implemented as an extension to the PROMISE streaming system [17], the contribution-level of a peer (i.e. contribution in terms of upstream bandwidth and possibly disk storage) is mapped to a score that determines the relative ranking of that peer in the system. Peers with higher rank are given more flexibility in peer selection by requiring that participants always accommodate service requests from a higher-ranking counterpart; thus, more generous contributors have a wider range of choices available to them when selecting their upstream parent(s). Given that the quality of a streaming session is very much dependent on the characteristics of streaming sources (e.g. upstream capacity, offered rate, availability) as well as the dynamics of the network paths from these sources to the receiver (e.g. packet loss rate, overlap of multiple paths from senders to the receiver, available bandwidth), this greater power in selection translates into higher-quality reception.

In this scheme, it is assumed that peers are strategic and so choose their contribution-level $x_i = x_i*$ so as to optimize an individual utility function $U_i(x_i)$:

$$U_i(x_i) = a_iQ_i(R_i(S_i(x_i))) - b_iC_i(x_i)$$

Here, x_i is some abstract characterization of contribution-level, $C_i(x_i)$ and $Q_i(R_i(S_i(x_i))$ represent the cost incurred and quality of service enjoyed, respectively, by peer i with contribution-level x_i and a_i and b_i are appropriate scaling constants. With the ability to compute the function above, a peer can determine its optimal contribution-level, thus enabling operation of the proposed incentive mechanism – the specifics of the calculation of $U_i(x_i)$ are provided below.

$C_i(x_i)$ can be any monotonically non-decreasing representation of cost versus contribution-level – for example, $C_i(x_i) = c_T\, T_i\, D_i$, where c_T is a unit transmission cost and T_i and D_i are the transmission rate and duration, respectively, of i. $S_i(x_i)$ must be a similarly non-decreasing function that maps x_i to a score – if $B_{in,i}$ and $B_{out,i}$ represent the number of bytes uploaded and downloaded, respectively, by peer i, them some legitimate options include: $S_i(x_i) = B_{out,i}$, $S_i(x_i) = B_{out,i} - B_{in,i}$, or $S_i(x_i) = B_{out,i} / B_{in,i}$. Scores may also be subject to a periodic decay factor, thus encouraging consistently good behavior.

$R_i(S_i(x_i))$ is the cumulative distribution function (cdf) of scores in the system, mapping the score of peer i to a percentile rank in the range [0,1]. In the absence of global information, a peer must approximate R_i by sampling a sufficiently large number of user scores – perhaps gathered through the passive monitoring of service requests containing score information. In [15] it is shown that the number of samples required to obtain reliable transmissions is bound by the variance of scores and not population size, thus allowing for a manageable sampling overhead even with large populations.

Finally, $Q_i(R_i(S_i(x_i))$ maps the percentile rank of peer i to the expected quality of service (QoS), quantified as a number in the range [0, 1], representing the fraction of all packets that are received on time. While the exact specification of this function may vary, it must always satisfy three conditions to serve its purpose of allowing a node to determine its optimal contribution-level. First, it should asymptotically approach Q_{MAX}, representing the highest level of quality that may be delivered by the system. Second, it should be monotonically non-decreasing in percentile rank and therefore, user score as well. Finally, it should have a non-negative initial value $Q_{BE} = Q_i(R_i(S_i(x_i) = 0))$, representing the "best-effort" service received by free-riders or newcomers that have not yet had the opportunity to accumulate a score. Note that this best-effort QoS level is entirely based on whatever service idle peers are willing to provide out of "charity" for users with scores and therefore ranks, less than their own. In [15], $Q_i(R_i(S_i(x_i))$ is defined as a simple linear function fitting empirical observations of quality versus percentile rank.

In [15], both simulation and wide-area experiments confirm the effectiveness of the approach above. The association of contribution-level and utility imposed by the incentive mechanism ensures greater participation by strategic peers, thereby increasing the capacity of the broadcast overlay and reducing the disproportionate burden otherwise placed on the small number of naturally occurring altruistic peers.

Nevertheless, a serious limitation of this work is its reliance on an implicit assumption that participants will not attempt to unfairly benefit by tampering with the client-side software that implements the incentive mechanism. Specifically, it would be trivial for a rogue client to bypass the mechanism altogether by reporting inflated scores, or simply denying service requests from higher-ranking peers. While [15] attempts to address this issue with such suggestions as weighing the information provided by another peer based on the extent to which that peer is trusted, or making use of a security infrastructure such as EigenTrust [21], the subjectivity of these solutions renders them inexact at best.

2.2 Taxation and Altruism

Another mechanism imposing a relationship between higher contribution levels and superior throughput is introduced in [8]. This approach is based on a metaphor of taxation – in effect, the bit-rate r_i received by a peer i is interpreted as an "income" on which a "tax" of f_i - the upstream bandwidth i contributes to the overlay – must be paid. By imposing a taxation scheme such that $f_i > r_i$ for all peers with sufficient upstream capacity, resource-rich peers effectively contribute surplus "public wealth" (i.e. excess bandwidth) that can then be redistributed to (i.e. consumed by) resource-poor peers. In effect, cooperative behavior is achieved and social welfare (i.e. system-wide utility) is maximized.

A simple linear tax schedule is used - this schedule is characterized by two parameters: (1) a tax rate specified by the publisher, t and (2) the demogrant, G, a parameter dynamically inferred from the peer environment that represents the minimum bit-rate all peers receive regardless of their contribution. The tax rate t is used in the formula $r_i^+ = f_i/t$ that specifies the entitled bandwidth r_i^+ of a peer i based on its contribution f_i to the broadcast overlay. Essentially, r_i^+ is the downstream bandwidth that peer i has rightfully "earned" by contributing a forwarding bandwidth of f_i. Equivalently - using the taxation metaphor - a peer with an entitled income of r_i^+ must pay a tax of f_i. Note that when $t > 1, f_i > r_i^+$ when $f_i > 0$ and $f_i \geq r_i^+$ when $f_i \geq 0$. Thus, assuming we have N peers numbered 1 to N, at least one of which contributes $f_i > 0$, we have:

$$\sum_{i=1}^{N} f_i - \sum_{i=1}^{N} r_i^+ > 0$$

This non-zero difference, called the "demogrant pool", corresponds to bandwidth that has been contributed to the broadcast overlay but is not "used up" in providing peers with their entitled bandwidth. Once all entitled bandwidth has been allocated, it is this surplus that is evenly distributed among all peers in the form of the demogrant G.

While the bandwidth allocation method for the proposed taxation scheme is conceptually defined above, an actual distributed protocol is needed to implement it. The protocol used in [8] is based on a modified version of the multiple disjoint tree technique introduced in [4, 27]. In the original specification of this technique, the source splits the stream into k equal "stripes" and multicasts each using a separate tree. These k trees, in turn, are formed by having each peer select one tree at random and joining it as an interior node and then joining the remaining $k - 1$ trees as a leaf node. In this way, each peer contributes forwarding bandwidth in the one tree where it is an interior node and receives bandwidth from all of the trees it joins. Assuming that each stripe has a bit-rate of v, a peer i can contribute f_i simply by having a fanout of f_i/v in the tree where it acts as an interior node. Similarly, to receive a bit-rate of r_i, it must join r_i/v trees.

Unfortunately, the basic multiple disjoint tree approach is not adequate on its own. While a peer i can determine f_i based on its self-interest and can easily derive r_i^+ from f_i using the tax rate t, the value of r_i depends on the demogrant G, which is a dynamically changing parameter based on global information. In order to obtain G and thus compute r_i and determine the number of trees (r_i/v) to join, the protocol must be extended to infer G in a distributed, online fashion.

In [8], two techniques are employed for the determination of G: (1) priority and (2) preemption. The first simply consists of each peer assigning a priority value for each of the trees it joins, with the first r_i^+/v joins (i.e. those that i uses to claim its entitled bandwidth) being labeled with the highest priority (priority = 0) and all subsequent joins (i.e. those used to claim i's share G of the demogrant pool) being labeled with decreasing priority (priority = 1, 2 and so on). The second technique, preemption, ensures that a join request of higher priority is never rejected at the expense of a request with lower priority. Specifically, if the fanout bound of an interior node is reached, a peer with higher priority can preempt the service of one with lower priority, thus ensuring that peers accepted into the tree always have a higher priority than those rejected. In this way, we guarantee that (1) all entitled bandwidth is claimed before demogrant (as a join request of priority 1 cannot be accepted at the expense of one of priority 0) and (2) that the demogrant pool is distributed evenly (as a join of priority $n + 1$, where $n + 1 > 1$, cannot be accepted at the expense of one of priority n).

While extensive simulation experiments in [8] bear out the effectiveness of this approach in ensuring cooperative behavior, it nevertheless suffers from the same contradiction as the rank based peer selection mechanism discussed in the last section. Specifically, it is assumed that a peer i will adhere to the rules of the taxation mechanism, joining only r_i/v trees (where $r_i = r_i^+ + G$) and contributing a forwarding bandwidth of f_i, by serving as an interior node with a fanout of f_i/v in one multicast tree. Unfortunately, there is nothing preventing a participant from achieving a very high utility by using a peer that implements a malicious variant of the protocol in which all k trees are joined without contributing any bandwidth.

2.3 Tit-for-Tat Punishment

In [26], a "tit-for-tat" strategy, in which uncooperative behavior is discouraged by identifying and directly retaliating against free-riders, is introduced and represents a first serious attempt at a truly incentives-compatible peer-to-peer streaming system, free of any reliance on the assumption of tamper-proof software. This scheme employs a number of mechanisms that allow a peer to identify free-riders, while only relying on its own first-hand observations. By using these mechanisms alongside periodic, random multicast tree reconstructions in which a peer A downstream of a peer B in the current tree will, with very high likelihood, be upstream of B in some future tree, it is ensured that A will be in a position to retaliate against B by providing it with a degraded quality of service QoS, should it identify B as a free-rider. Although the techniques for identifying uncooperative peers and retaliating against them are presented as an extension of SplitStream [4] - an application level multicast service that is implemented over the Pastry [34] P2P routing substrate and makes use of the same multiple disjoint tree structure discussed in Section 2.2 - they can easily be applied to a broad selection of tree-based multicast systems.

Four mechanisms, to be used in combination, are proposed for the identification of free-riders: (1) debt maintenance, (2) parental availability tracking, (3) reciprocal requests and (4) ancestor rating. Again, in all of these, difficult issues of trust are avoided by only requiring a peer to base its judgments on its own first-hand observations. In debt-maintenance, whenever a peer A forwards a packet to a peer B, such that B is an immediate child of A in a multicast tree, both nodes note that B owes A a debt of one packet. As noted in [26], if trees are constructed in a random fashion, A and B can expect to be the parent of the other with equal frequency and under such circumstances, the expected average accumulated debt will vary with the square root of the number of tree reconstructions. The "debt level" metric tracked by both of these peers is thus defined as the total accumulated debt divided by the square root of the number of reconstructions. If the debt level that *A* has for *B*, or vice versa, increases beyond a certain experimentally determined threshold, *A* can identify *B* as a likely free-rider and deny it service until this debt is reduced.

The next mechanism, parental availability tracking, allows a peer *A* to classify a peer *B* as either a normal or free-riding node based on *B*'s past history of accepting or rejecting *A*'s requests to be a child of *B*. When a peer *A*'s request for service from a prospective parent *B* is rejected, it may be due to the legitimate reason that *B* is already serving the maximum number of children that can be supported by its upstream capacity. If, however, after numerous tree reconstructions *B* demonstrates an unduly high frequency of such rejections, *A* can identify it as a free-rider and deny it service. Note that what constitutes an "unduly high frequency" of rejections is very much a function of the multicast tree construction scheme used.

Reciprocal requests can be viewed as an extension of parental availability tracking. Assuming that tree reconstructions select new topologies in a random fashion, two well-behaved peers *A* and *B* would expect to be parents of each other with equal frequency. Thus, if a large disparity were to develop over time in the frequency with which *A* was the parent of *B*, versus the frequency of the reverse, the disadvantaged peer (say *A*) could reasonably expect that the other was a free-rider. Rather than denying the suspect node service, however, *A* could test its hypothesis by at some point deviating from the standard join protocol and explicitly requesting to be a child of *B*. If *B* were to reject *A*, then it could be identified as a free-rider with high-probability, otherwise, *A* may assume that the earlier imbalance in its relationship with *B* occurred by chance.

In ancestor rating, each peer maintains a confidence value for every peer that has ever served as an ancestor for it in a multicast tree. When a node successfully receives a packet, it increments its confidence level in each of the peers on the path up to and including the root of the given tree. Should the peer fail to receive a packet for a stream within the required window of time, its confidence in each of the peers is reduced.

Furthermore, these confidence values are periodically decayed, thus encouraging peers to consistently behave in a cooperative fashion.

Clearly, the form of "collective punishment" used in ancestor rating will falsely assign blame to normal nodes in the event that they are on a path containing a free-rider, or along which any number of "natural" packet losses have occurred. Nevertheless, assuming that free-riders do not make up an overwhelming fraction of the participants and that packet losses are relatively infrequent, this effect would be corrected for after the cooperative behavior of normal nodes is observed with subsequent reconstructions. The confidence levels for free-riders, however, would be consistently degraded thus allowing for them to be targeted.

Simulation experiments in [26] demonstrate that a combination of the incentives-compatible mechanisms outlined above can be used to identify and punish free-riders – after a sufficient number of tree reconstructions, free-riders were able to receive only 10% of the published stream. In spite of these results, the effectiveness of this approach remains questionable for several reasons. First, the simulations exposed very slow-responsiveness to free-riding behavior – the bandwidth consumed by selfish peers was only gradually reduced to 10% of the published stream after hundreds of tree reconstructions. While this problem may be somewhat alleviated by increasing the frequency of tree-reconstructions, this would significantly increase protocol overhead – illustrating an unfortunate inherent trade-off. Second, even though uncooperative peers were only able to receive 10% of the published stream, this may represent a significant fraction of the total capacity of the system if the proportion of free-riding peers is sufficiently large. Without some provision for ensuring that a free-rider could not attempt to join under several separate pseudonyms, it is also conceivable that the small fractions of the stream received under multiple pseudonyms could be pieced together to obtain a reasonable QoS-level – thereby circumventing the incentive-mechanism altogether. Third, an overly simplistic model of free-riding behavior was employed – it was assumed that free-riders are consistently uncooperative. A more sophisticated offender that only sporadically misbehaved would able to evade detection by the mechanisms above to at least some degree. Finally, the successful results generated were for specifically tuned values of the parameters characterizing the mechanisms used – it was by no means demonstrated, however, that a single, static set of values for these parameters could be effective across a range of streaming scenarios.

2.4 Market Quotes

In [35], a "Bazaar Framework" is introduced - as with tit-for-tat punishment and unlike some earlier approaches that attempt to address the problem of free-riding through the imposition of artificial, rule-based mechanisms, this represents a second serious attempt at an incentives-compatible solution. In this work, it is observed that natural incentives for cooperation exist for selfish, utility-maximizing peers in the peer-to-peer media streaming domain and these may exploited. Consider, for example, a system with a "root" streaming server *R* , with an upstream capacity of 400kbps and four client peers *A*, *B*, *C* and *D*, each with an upstream capacity of 200kbps and a downstream capacity of 400kbps. If each of the client peers behaves in an uncooperative fashion – i.e. connects directly to *R* and refuses to forward any streaming data – then *R*'s upstream capacity will be divided four ways and each client will receive data at only 100kbps. By cooperating, however, these peers can double the throughput they receive – *R* can stream to *A* and *B* at 200kbps and these clients, in turn, can forward 200kbps streams to *C* and *D*, respectively. Clearly, in a system with sufficient competition for a server's upstream capacity, selfish peers should have an incentive to cooperate and peers will seek to optimize their individual utilities by making whatever tradeoffs are needed between maximizing received throughput and minimizing the upstream bandwidth used.

The Bazaar Framework consists of three elements: (i) the root, (ii) the client peers and (iii) the Boot Strap Entity (BSE). The root is the server that publishes the stream – it contributes a fixed portion of its upstream capacity that is evenly divided among all clients directly connected to it. The client peers are the consumers of the streaming content and may either be altruistic, in which case they use as much of their upstream capacity as possible to forward streaming data, or selfish, in which case they try to maximize some utility function that increases with the throughput received and decreases with the fraction of upstream bandwidth donated to the streaming overlay. Finally, the BSE is a central repository of information about the nodes in the system that bootstraps peers joining the system. The BSE maintains a quote repository that contains a "market-quote" for each peer in the system – this quote is "advertised" by the peer to the BSE when it joins and consists of three elements: (i) some peer identifier (e.g. an IP address and port number), (ii) the bandwidth that the peer is willing and able to provide to a client that joins as its immediate child in the overlay and (iii) some indication of the latency incurred along the overlay from the root to the peer.

In the Bazaar framework, a streaming session is initiated by the root R advertising its market quote (R_{id}, R_{bw}, R_{lat}) to the BSE – here R_{id} is the identifier for R, R_{bw} is the upstream bandwidth R is willing to contribute to the overlay and R_{lat} is the associated latency. Clearly, $R_{lat} = 0$ as R is the root itself and for the sake of discussion we will assume that $R_{bw} = 400$kbps. When a client peer A joins the overlay, it will query the BSE for the set of market quotes and receive the entry (R_{id}, R_{bw}, R_{lat}), as this is the only entry in the system, A will send a join request to R and then begin receiving the stream at the advertised rate of $R_{bw} = 400$kbps. As R now has a child, it must publish a revised market quote to the BSE – the quote repository will now contain the advertised rate of $R_{bw} = 200$kbps, as the next peer that joins as a child of R will have to split the 400kbps that R is willing to contribute with A. Once A has begun receiving streaming data, it may or may not choose to publish a market quote (A_{id}, A_{bw}, A_{lat}) of its own to the BSE. Although A's behavior is entirely determined by whatever utility function best characterizes its circumstances, it presumably has a strong incentive to publish a market quote that is competitive with that published by R, as should a new peer elect to join as a child of R instead of A, the throughput received by A will be sharply reduced from 400kbps to 200kbps. As A has to compensate for the fact that $A_{lat} > R_{lat}$, a competitive quote will contain a value for A_{bw} such that $A_{bw} > R_{bw}$. Nevertheless, A_{bw} may be substantially less than A's upstream capacity – if A is a selfish peer, it will try to select the lowest possible value of A_{bw} sufficient to discourage any newcomer from splitting the capacity of its parent.

The scheme outlined above is supplemented by a "shuffle" mechanism that allows for a peer willing to increase its offered bandwidth to initiate an appropriate adjustment of the overlay. Consider a streaming session in which the root peer *R* is dividing a 400kbps upstream capacity evenly among two client peers *A* and *B*, each initially deciding to forward no data. Realizing that it can receive *B*'s share of *R*'s bandwidth, *A* can initiate a shuffle operation by sending a market quote to *R* that it is specifically directed to *B* - *R* will then forward this offer to *B*. This local quote will have to contain a value A_{bw} that is sufficiently higher than the 200kbps currently enjoyed by *B* in order to be palatable, say A_{bw} = 300kbps. If *B* determines that it can increase its utility by accepting the proposed arrangement it will then detach from *R* and rejoin the overlay as a child of *A*, thereby completing the shuffle. Now, *A* will receive 400kbps from *R* and forward 300kbps to *B*, increasing the quality enjoyed by both clients. While the Bazaar framework described in [35] implements the Shuffle-1 operation described above – i.e. a shuffle that involves clients on the same level of the streaming tree – a Shuffle-*k* operation, involving multiple peers spanning *k* levels of the tree is conceivable.

Unlike the tit-for-tat approach, the Bazaar framework seems to effectively ensure cooperation in the presence of selfish peers. Simulation experiments in [35] demonstrate that high-levels of cooperation and thus throughput and system-wide utility, are achieved across populations composed of varying mixtures of selfish and altruistic peers, possessing a range of upstream capacities and making use of several permutations of join order. A limitation of the framework, however, is that its dependence on the use of a single tree leaves it susceptible to a high-degree of volatility in the throughput received by participating peers. As the upstream capacity of a peer is evenly divided among its children, a join operation will always have a significant impact on any siblings. This problem is magnified by the fact that the ability of a peer to provide the bit rate promised in its market quote may directly depend on the ability of all of its ancestors in the overlay tree to meet the QoS parameters promised in their quotes – thus, a local degradation can ripple down the sub-tree rooted at an affected peer, without any mechanism in place for cleanly restoring the overlay.

CHAPTER 3

Design

In this chapter the design of an incentives-compatible peer-to-peer media streaming protocol is presented – this approach attempts to address the shortcomings of earlier schemes, as identified in the preceding survey. The protocol developed is based on a novel extension of the CoopNet [27] system, a peer-to-peer media streaming system that employs the multiple-disjoint tree approach referenced in Chapter 2. Specifically, in CoopNet, a content provider divides its media stream into multiple "stripes", or "descriptions", each consuming an equal amount of bandwidth and disseminated along a separate multicast tree composed of client peers. The membership and topology of each multicast tree is determined by a centralized algorithm run by the content provider and the number of stripes – or equivalently, the quality of the media stream - received by a client, depends on the number of multicast trees that it participates in. Recognizing that the tree management algorithm used thus has complete control over the quality of service received by each peer, this work replaces the simple algorithm employed by CoopNet with one that enforces a clear correlation between the contribution level of a peer and the quality of service it enjoys, thereby incentivizing cooperative behavior and penalizing free-riders.

The remainder of this chapter is organized as follows. In Sections 3.1 and 3.2, respectively, the notions of striped transmission and centralized tree management are discussed – as they form the basis of the CoopNet system they are central to the approach developed in this work and so, are presented in further depth here. Finally, in Section 3.3, a tree management algorithm that relates quality of service and contribution in an incentives-compatible fashion is introduced.

3.1 Striped Transmission

To present the central notion of striped transmission in more concrete terms, the particular "Multiple Description Coding" (MDC) scheme employed by CoopNet is presented here in further detail. As shown in Figure 3-1 (adapted from Figure 3 of [27]) the CoopNet MDC architecture consists of four logical components: the Prioritizer, Tree Manager, Optimizer and Packetizer.

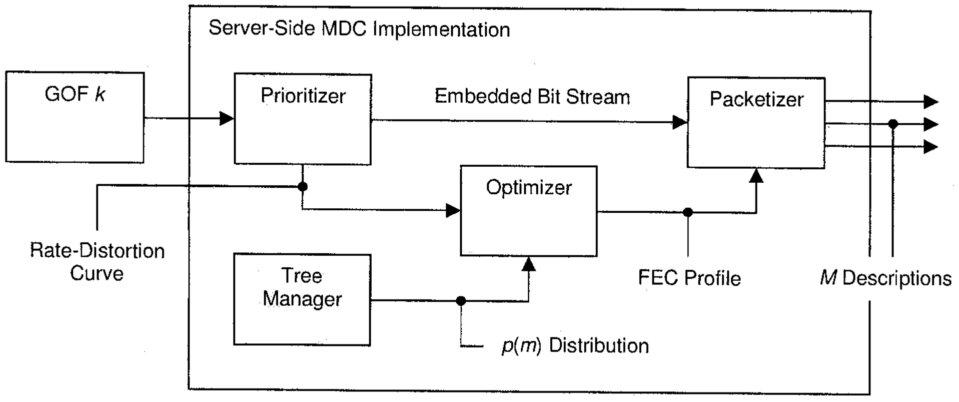

Figure 3-1: Example MDC Architecture

The points below summarize the operation of these components in producing separate descriptions from a single media stream. For the sake of discussion, it will be assumed that a video stream – i.e. a sequence of frames – is being processed.

Step 1: The prioritizer consumes the input stream from an appropriate codec, ingesting one "group of frames" (GOF). Each GOF represents a contiguous segment of the video of duration T – typically $T = 1$ second – and does not overlap with any other GOF.

Step 2: The prioritizer partitions the GOF into a collection of data units. Each data unit is annotated with rate-distortion information that specifies how much the information in that particular unit can contribute to a reduction in distortion when reconstructing the GOF. This rate-distortion information is then used to sort the data units in order of decreasing importance – i.e. those units that most contribute to a reduction of distortion appear first. The resulting sequence of bits is called an "embedded bit stream" and is sent to the packetizer. The prioritizer also uses the rate-distortion information to produce a "rate-distortion" (RD) curve that is fed to the optimizer. This RD curve essentially specifies a function $D(R)$ that gives the distortion that would result if only the first R bits of the embedded bit stream were available to reconstruct the original GOF. Thus if $0 = R_0 \leq R_1 \leq \ldots \leq R_M$, where R_M is the total number of bits in the embedded bit stream, then $D(R_0) \geq D(R_1) \geq \ldots \geq D(R_M)$.

Step 3: The tree manager computes the function $p(m)$, which gives the probability that a client will receive m of the complete set of M descriptions for a GOF and feeds this function to the optimizer. This probability density function is determined and continuously updated by the tree manager using statistics reported by the clients. Over a streaming session, each client maintains a histogram of the number of descriptions it receives per GOF and periodically reports these statistics to its parent in a tree designated for this purpose. The parent then adds the histograms received from its children to its own and passes the result to its parent and so on. By recursively continuing this process from the leaves to the root peer running the tree manager, a histogram incorporating statistics for all of the clients is available and can simply be normalized to obtain $p(m)$.

Step 4: The optimizer uses the RD curve and the probability density function $p(m)$ to determine the optimal "Forward Error Correction (FEC) profile" for the GOF. The FEC profile consists of a set of points $R_0, R_1, \ldots R_M$, where $R_0 < R_1 < \ldots R_M$

and M is the number of descriptions to be used, that partition the embedded bit stream. This partitioning is such that any bits in the range $[R_{i-1}, R_i)$ are mapped to i source blocks and $M - i$ FEC blocks, generated using Reed-Solomon encoding [41]. By distributing these source and FEC blocks across M packets, any m of these M packets can be used to obtain the first R_m bits of the embedded bit stream. This is illustrated in Figure 3-2 (adapted from Figure 2 in [27]). Note that with the probability function $p(m)$ from the optimizer and the RD curve providing the rate-distortion function $D(R)$, the optimal values of R_0, R_1, ... R_M can easily be computed as expected distortion is given by the formula:

$$\sum_{m=0}^{M} p(m)D(R_m)$$

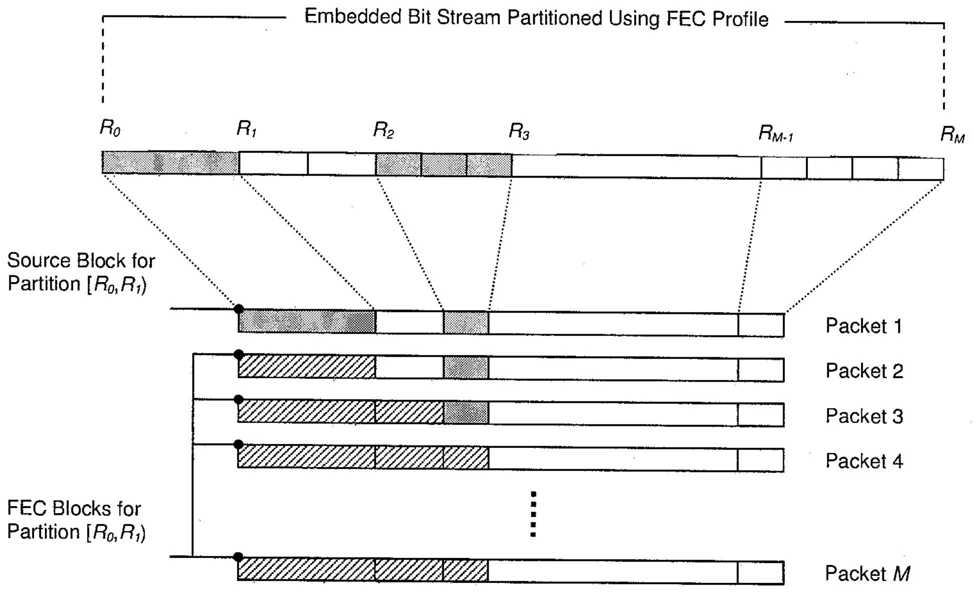
Embedded Bit Stream Partitioned Using FEC Profile
R_0
R_1
R_2
R_3
R_{M-1}
R_M
Source Block for Partition $[R_0,R_1)$
FEC Blocks for Partition $[R_0,R_1)$
Packet 1
Packet 2
Packet 3
Packet 4
Packet M

Figure 3-2: Packetization of Embedded Bit Stream with FEC Data

Step 5: Using the embedded bit stream and the FEC profile, the optimizer produces M packets for the GOF, writing the GOF number in the header of each. Each of these M "descriptions" is then distributed to the participating clients over the appropriate end-host multicast trees (different descriptions may be disseminated over different trees).

Step 6: Listening on the end-host multicast trees for the descriptions, a client will receive $m \leq M$ packets for the GOF and FEC-decode them to obtain the first R_m bits of the embedded bit stream. With this information, the client can then reconstruct the GOF with a distortion of $D(R_{\mathrm{m}})$.

Step 7: Repeat from Step 1 until there are no further GOFs to be ingested.

Although many alternatives to the striped transmission scheme outlined above are available, as surveyed in [13], the protocol proposed in this work is agnostic to the mechanism used, so long as two requirements are satisfied: (i) a client should be able to decode any subset of stripes and (ii) the quality enjoyed by a client should strictly be a function of the *number* of stripes received.

As a final aside, it should be noted that in addition to enabling the protocol proposed in this work by allowing the quality of service a peer receives to be controlled by publishing a particular number of descriptions to it, striped transmission offers several other advantages over the use of a single media stream. First, by distributing a transmission across several descriptions along separate multicast trees and allowing a client peer to at least partially reconstruct the transmission with any subset of these descriptions, striped transmission offers greater robustness in the face of peer transience. With a single multicast tree, a node failure or departure will result in all clients in the affected sub-tree losing reception altogether until the overlay can be repaired. With striped transmission along several trees, however, only the reception of those descriptions for which the lost peer was an interior node in the corresponding multicast tree will be affected – this effect is considerably mitigated by the fact that (a) the reception of the remaining descriptions will still allow for at least part of the original transmission to be recovered and (b) an intelligently designed tree management algorithm will ensure that each peer is an interior node in as few multicast trees as possible.

Second, striped transmission allows for network load to be more evenly distributed among participants in the overlay. As noted in [27], a single, balanced multicast tree with fanout f and height h will have f^h leaf nodes but $f^h -1 / f^h - 1$ interior nodes and thus, the fraction of leaf nodes to interior nodes will increase linearly with f. As it is only the interior nodes that contribute upstream bandwidth in an end-host multicast scheme, this imbalance places a disproportionate burden on a few nodes and limits the capacity of the system. The use of multiple trees, however, will of course alleviate this burden by allowing a greater number of clients to participate as interior nodes.

Finally, striped transmission allows the upstream capacities of participating peers to be better leveraged in spite of their considerable heterogeneity in this regard - while a single large stream will prevent peers with insufficient upstream capacity from contributing any bandwidth to the overlay, a transmission divided into sufficiently "narrow" stripes will allow even resource-poor peers to at least forward some subset of stripes.

3.2 Centralized Tree Management

As discussed earlier, both CoopNet and the extension presented in this work, make use of centralized tree management. In particular, the root peer is responsible for determining the topologies of the dissemination trees to be used and ensuring that participating peers are efficiently organized into an overlay reflecting this structure. A key advantage of centralized tree management is the control it affords the content provider - by giving the root peer complete control over the topology of the streaming overlay, it is empowered to determine the quality of service received by each participating peer; further, with knowledge of the topology used, the root has an omniscient view of the levels of bandwidth contribution and consumption for each of the various participants. Recall, that the effective throughput received by a peer is determined by the number of stripes that it receives and hence, the number of dissemination trees it is a member of - also, the reliability with which a peer receives these stripes is determined by the number and stability of its ancestors in these various trees. Likewise, the bandwidth contributed by a peer is determined by the number of children it services in the overlay. This centralized control and oversight forms the basis of this work – a tree management algorithm that observes cooperative behavior and rewards it with topologies more favorable to contributors. Another key advantage of centralized tree management is its simplicity, which ensures that the overlay can be efficiently maintained; as will be later discussed, a graceful join or leave operation only requires that the arriving or departing node inform the root and that the root appropriately updates other affected nodes, all of which can be accomplished within two network round-trip times (RTTs) [27].

Two arguments against the centralized design proposed are that placing the entire management burden on the root inherently limits scalability and that, in this way, the root constitutes a single point of failure [27] – fortunately, these concerns are easily addressed. Indeed, the system is limited by the bandwidth, CPU and memory resources of the root on which the tree management algorithm is run. In terms of bandwidth, however, this central bottleneck only applies to a relatively insignificant load of control traffic used to manipulate the topology of the overlay – with respect to the bandwidth requirements of the actual media stream being distributed, however, the system still scales with demand. In terms of CPU and memory, the extent to which a central bottleneck raises scalability concerns depends entirely on the tree management algorithm implemented. Experimental results for the algorithm implemented by CoopNet, however, indicate that a range of centralized schemes should be feasible – in [27], it was noted that simple 2GHz Mobile Pentium 4 could handle a population of over 10000 nodes, with 400 joins and leaves per second. Finally, the latter argument, identifying the root as a single point of failure, is not of concern as given that the media stream will typically originate from the root alone, the system will suffer from a central point of failure irrespective of the tree management scheme used.

In any centralized approach, the root must support a set of basic primitives for constructing and maintaining the overlay as nodes join and leave the system – the subsections below detail the join, leave and repair operations implemented for the CoopNet framework. It should be emphasized that these primitives are agnostic to the underlying tree management algorithm employed – they can be used to implement any number of policies. For this reason, while originally implemented for the CoopNet system, these primitives are used by the system proposed in this work without modification.

3.2.1 Node Join

When a new peer wishes to join a streaming session, it sends a join request message to the root peer – i.e. the content provider. Using its tree management algorithm, the root can then determine which, if any, peers already participating in the overlay will serve as parents of the new node; of course, the peer will be assigned *at most* one parent in the dissemination tree for each stripe. Figure 3-3 provides an example of this operation.

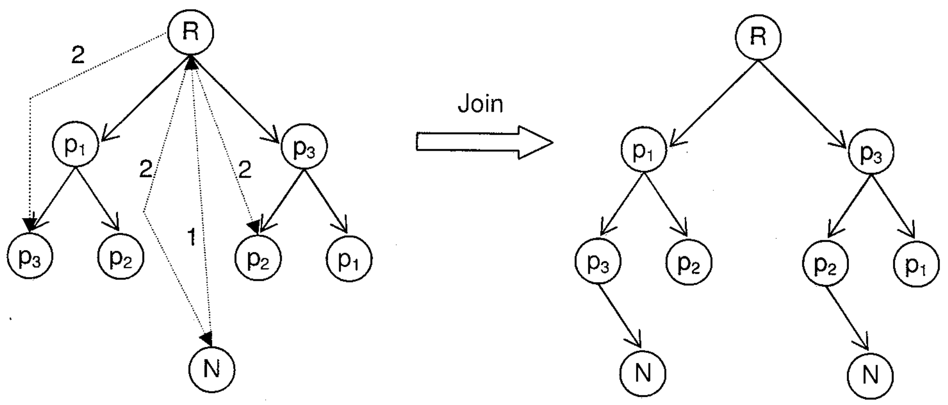

Figure 3-3: Node Join Operation

Here, the root peer R disseminates two stripes – one along the multicast tree rooted at p_1 and the second along the tree rooted at p_3. A new peer N wishing to join the streaming session will start by sending a join request to R – shown with the thin arrow from N to R labeled 1. Through its tree management algorithm, R determines that it would like N to join the tree for the first stripe with p_3 as its parent and to join the tree for the second stripe with p_1 as its parent. To modify the overlay accordingly, R sends a message to p_3 indicating that it must pass the first stripe to N, a message to p_1 indicating that it is to forward the second stripe to N and finally a message to N indicating that it will be receiving these stripes from p_1 and p_3 – this is shown with the thin arrows emanating from R labeled 2. The resulting overlay is shown on the right side of Figure 3-3, with N having joined both multicast trees. Note that although shown in the example above, it is not mandatory for a new peer to join a multicast tree as a leaf. Should the tree management algorithm decide to make it an interior node, the root will notify the new node and its chosen parents as before, however, it will also have to: tell the new node to serve its new children, inform children of the identities of any new parents selected for them and finally, inform parents of the identities of any children moved.

3.2.2 *Graceful Node Leave*

When a peer wishes to leave a streaming session, it will ideally do so in a controlled fashion by sending a leave request to the root peer. Again, using its tree management algorithm, the root peer will determine the new parents of any children being orphaned by the departing node and then notify the affected peers of the new parent-child relationships to be observed. To minimize disruption to downstream nodes, the departing peer will ideally continue to participate in the overlay until its parents cease to forward traffic to it, at the instructions of the root. Figure 3-4 provides an example of this operation. In this scenario, we again have an overlay with two multicast trees disseminating two stripes. Should peer p_2 wish to leave the overlay, it will send a leave request to the root, R – shown with the thin arrow labeled 1. Here, both p_1 and p_3 are affected by the departure of p_2 – p_1 loses p_2 as a child in the first dissemination tree and both p_1 and p_3 lose p_2 as a parent in the second tree. To accommodate these changes, R communicates the rearranged overlay topology determined by its tree management algorithm to p_1 and p_3 – shown by the thin arrows labeled 2. Specifically, R tells p_1 to stop forwarding the first stripe to p_2 and further, that it will now receive the second stripe from p_3. R also informs p_3 that it will now receive the second stripe directly from the root and instructs it to forward this stream on to p_1. The end result of this reconfiguration is shown on the right side of Figure 3-4.

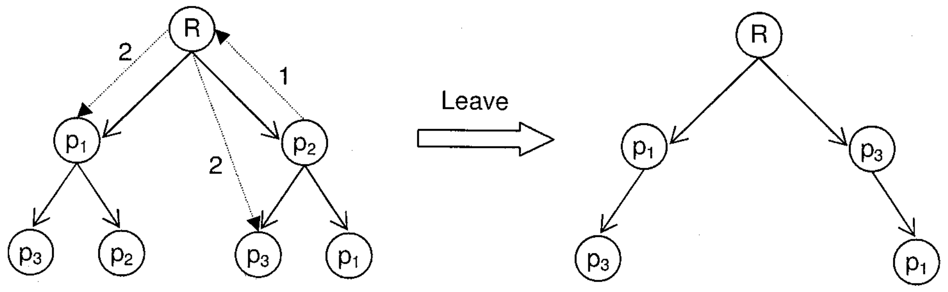

Figure 3-4: Graceful Node Leave Operation

3.2.3 Ungraceful Node Leave

It is possible that a peer will cease to participate in a streaming session without providing any prior notification – legitimate reasons for such an "ungraceful leave" may include anything from a sudden loss of network connectivity or crash of the end-host software, to unexpected congestion arising from contention with other applications. To handle these situations, each peer monitors the loss rate for each of its incoming stripes by inspecting the packet sequence numbers on each and noting any gaps or stoppage. Should the loss-rate for a particular stream exceed an acceptable threshold, the peer can assume that at least one of its ancestors has effectively departed or fallen out of communication. In this case, the peer will check with its current parent to see if it is also experiencing a loss of service. If the parent responds indicating that it is, the peer waits for its parent to resolve the situation – alternatively, if the parent fails to respond to the peer, or responds without any restoration of service within a timely manner, the complaining peer will contact the root directly, requesting a new parent. In this scheme, while every peer in a sub-tree affected by an ungraceful leave will query its parent for loss rate information, only the root of each sub-tree will fail to receive a response and so, only these root peers will directly contact the content provider root *R*. At this point, *R* can select new parents to restore service to the affected sub-trees and note the identity of the peer that ceased to forward data without warning - should this information be a useful input to the tree management algorithm employed. A simple example of the procedure outlined above is illustrated in Figure 3-5.

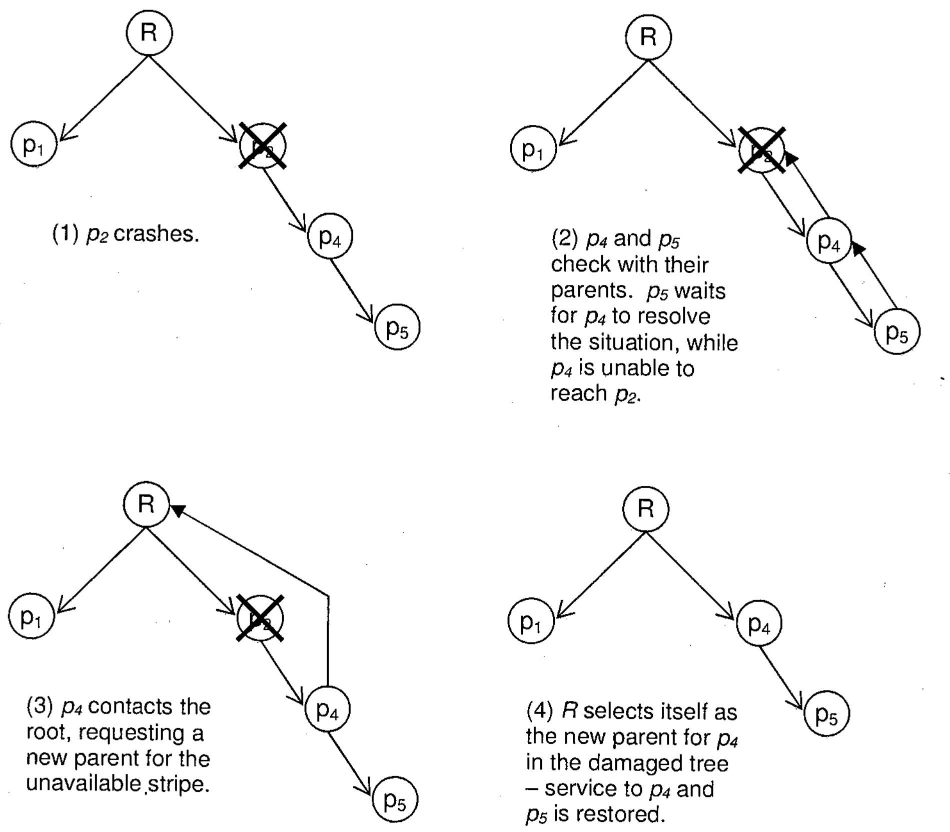
R
p1
p2
p4
p5
(1) p2 crashes.
(2) p4 and p5 check with their parents. p5 waits for p4 to resolve the situation, while p4 is unable to reach p2.
(3) p4 contacts the root, requesting a new parent for the unavailable stripe.
(4) R selects itself as the new parent for p4 in the damaged tree – service to p4 and p5 is restored.

Figure 3-5: Ungraceful Node Leave Operation

In addition to the legitimate reasons for an ungraceful leave cited above, a peer that decides to start free-riding, by no longer forwarding some or all of the stripes that it was ordered to distribute by the root, will essentially have the same effect as an unexpected departure and can be handled in a similar fashion. To discourage false reports of delinquent nodes, however, the root may contact the alleged offender, instructing it to cease forwarding data to the child claiming that it is not receiving the expected stripe. If the root is unable to contact the alleged offender during step, it can assume that the peer has left the overlay and simply process it as a leave operation.

As a concluding aside, it should be emphasized that the node join and leave examples above are only intended to illustrate the concept of centralized tree management – whereby the content provider is supplied with the information needed to monitor membership of the overlay and issues directives to participating peers to control the overlay topology. These examples are not meant to suggest any particular tree management policy – that is entirely the domain of the tree management algorithm and the content provider is free to employ any suitable such algorithm and issue whatever directives it deems appropriate. Section 3.3 develops the algorithm proposed in this work as a means for achieving incentives compatible peer-to-peer media streaming.

3.3 Tree Management Algorithm

3.3.1 Design Goals

The algorithm used to determine the topology of any tree-based streaming overlay must address multiple, arguably disparate, design goals – the paragraphs below identify several desirable criteria considered when formulating the tree management scheme employed by OmniStream. It should be noted that the first five of these criteria are not unique to this work – rather, they are the requirements that were identified and used to drive the design of the CoopNet framework, on which the proposed scheme is based. These five criteria are concerned with ensuring high stability and low latency when distributing streaming data through a large and highly dynamic population of participating peers. The sixth and final criterion is concerned with the free-riding issue addressed by OmniStream's enhancement of CoopNet.

(1) *Limited tree heights.* The inherent unreliability of the participating end-hosts in any peer-to-peer media streaming framework is unavoidable and must be accounted for accordingly. In the typically conceived of scenario, peers participating in the overlay are likely to run on small, personal hardware platforms, under the independent administrative control of personal users. This leaves the overlay particularly susceptible to the unexpected "departure" of peers due to such possibilities as the improper shutdown of the streaming client application, failure at the hardware of software level, or loss of network connectivity. Moreover, as discussed at the outset, these personal users are likely to make use of "last-hop" broadband connectivity with a highly constrained upstream capacity. As the streaming client application may unexpectedly have to compete with other network services for this scarce upstream bandwidth (e.g. the personal "administrator" of the node initiates a large FTP upload), the problem of unreliable peer service is further exacerbated.

Given their high probability, disruptions at the end-host level are likely to have a significant impact on streaming quality – perhaps more so than failures in the network itself. To mitigate this concern, the overlay management algorithm must minimize the height(s) of the tree(s) constructed – in doing so, the number of end-hosts traversed from the root to a given peer is reduced and so too, therefore, is the likelihood of service interruption due to the failure of an upstream ancestor. Note that minimizing tree height is effectively equivalent to maximizing the "fan-out" at each peer – thus, shorter trees can be obtained by ensuring that each peer serves as many clients as possible.

(2) *Disjoint sets of ancestors.* In a striped transmission scheme making use of multiple trees, the set of ancestors a peer has in one tree should have as few members in common as possible with the set of ancestors that peer has in any other tree. The motivation for this disjointedness is once again, to mitigate the impact that the unexpected disruption of end-hosts may have on streaming quality. By ensuring that a peer p is an ancestor to a peer q in as few trees as possible, the number of stripes that q ceases to receive in case p unexpectedly

leaves the overlay is limited. Note that maximizing disjointedness is equivalent to limiting the number of trees in which any peer serves as an interior node (i.e. has client peers itself).

(3) *Scalability.* Clearly, the utility of a peer-to-peer media streaming infrastructure is a function of the number of concurrent clients it can effectively service. Any tree management algorithm must be able to accommodate large client populations and handle the significantly higher rates of node joins, leaves and failures associated with larger numbers of participating peers.

(4) *Minimized impact of join and leave operations.* Especially as the number of potential clients increases, it is reasonable to expect a peer-to-peer media streaming session to be an extremely dynamic environment, with frequent change to the overlay topology maintained. In such circumstances, the tree management scheme employed must minimize the delays peers must experience before joining or leaving the overlay, as well as the service disruptions caused by unexpected departures.

(5) *Close correlation between network and overly topologies.* By accounting for the topology of the underlying physical network, an overlay management algorithm can reduce the latency experienced by the participating peers – i.e. the delay in receiving streaming content from the root. For example, in a streaming session including peers situated in both North America and Europe, a tree can be constructed so as to minimize the number of trans-Atlantic hops a packet must make in traveling from the root to a peer. Another advantage of accounting for physical topology in this manner is that it allows for more efficient use of the underlying network. For example, if a single backbone link *m* was used in the application-level links between peer *a* and peer *b* and from peer *b* to peer *c*, then a single packet streaming from *a* to *b* and then onto *c* would unnecessarily traverse *m* twice.

(6) *Ability to ensure cooperative peer behavior.* To address the free-rider problem in the context of peer-to-peer media streaming, a tree management algorithm must enforce powerful incentives for the contribution of upstream bandwidth, as well as penalize the unreciprocated consumption of bandwidth. While the techniques of striped transmission and centralized tree-management applied by CoopNet address the first five design goals noted above, again, in this work, it is noted that these methods also provide the means by which the free-riding problem may be addressed. With centralized management, the root peer, or whichever node is responsible for executing the tree management algorithm, is able to maintain an omniscient view of the overlay network, carefully monitoring the contribution and consumption levels of each participant. With striped transmission, the quality of service received by each individual peer can be controlled by adjusting the number of trees the peer is granted membership to.

OmniStream, the extension of CoopNet developed in this work, applies these observations to ensure that peers receive a number of stripes commensurate with their individual contribution levels. The specific means by which OmniStream quantifies the net contribution of a peer over time is outlined in Section 3.3.2. Section 3.3.3 delineates the manner in which this information is applied to the construction of the various overlay trees used to disseminate stripes.

3.3.2 Quantifying Net Peer Contribution

As the root peer is responsible for constructing and maintaining the overlay as nodes join and leave the system, or unexpectedly fail, this root has complete knowledge of the overlay topology in place at any given time. In particular, for any given peer p, the root knows how many multicast trees p is a member of (i.e. how many stripes p is receiving, or equivalently, how much downstream bandwidth it is consuming), as well as how many children p has in each of those trees (i.e. how much upstream bandwidth p is contributing to the overlay). Thus, if $r_p(t)$ is used to denote the number of streams that p is receiving at time t and $f_p(t)$ denotes the number of streams that p forwards at time t, then, the net bandwidth contributed by p to the overlay at t, $b_p(t)$, is given by the trivial formula below, where k is the common bit-rate at which each stream is transmitted:

$$b_p(t) = k \cdot (f_p(t) - r_p(t))$$

Clearly, $b_p(t) > 0$ indicates that p is a net contributor to the system at time t, $b_p(t) = 0$ indicates that p contributes and consumes bandwidth in equal amounts and if $b_p(t) < 0$, then p is a net consumer.

While $b_p(t)$ characterizes the contribution-level of a peer at a given instant in time, it does not convey whether or not the peer has made a positive, negative or neutral contribution to the streaming infrastructure over its lifetime in the system – a separate metric, $C_p(t)$, is defined here for this purpose. The lifetime of a peer is simply the period for which it is known to the streaming infrastructure – it need not participate in any overlay over this time, of relevance is only that it is somehow registered and tracked. The time t_0 will be used to denote the time at which the lifetime of peer p begins. Subsequent to t_0, p is affected by a series of events, where an event is anything that may change $b_p(t)$ – that is, anything that may cause p to start or cease to receive or serve one or more streams (e.g. a node joining or leaving the overlay, failing or being affected by a failure upstream, etc.). Assuming that as of time t, the lifetime of p has been punctuated by m events, $t_1, \ldots, t_m$, where $t_0 \leq t_1 < \ldots < t_m \leq t$, will be used to indicate the points at which events 1, ..., m transpired, respectively. Noting that p has a contribution-level of $b_p(t_i)$ at time t_i and by definition, this does not change until event $i + 1$ occurs at time t_{i+1}, $C_p(t)$ may be defined by the following formula, where $\Delta t_i = t_i - t_{i-1}$:

$$C_p(t) = \sum_{i=1}^{m} b_p(t_{i-1})\Delta t_i + b_p(t_m)(t - t_m)$$

As $C_p(t)$ is effectively a product of bandwidth and time, it evaluates to a figure whose units are of data – thus, if $b_p(t)$ is given in bits-per-second (bps) and the various t_i are in seconds then $C_p(t)$ will provide the net contribution peer p has made to the streaming infrastructure over its lifetime in bits. This is best illustrated in Figure 3-6, below – where $C_p(t)$ is represented by the shaded area.

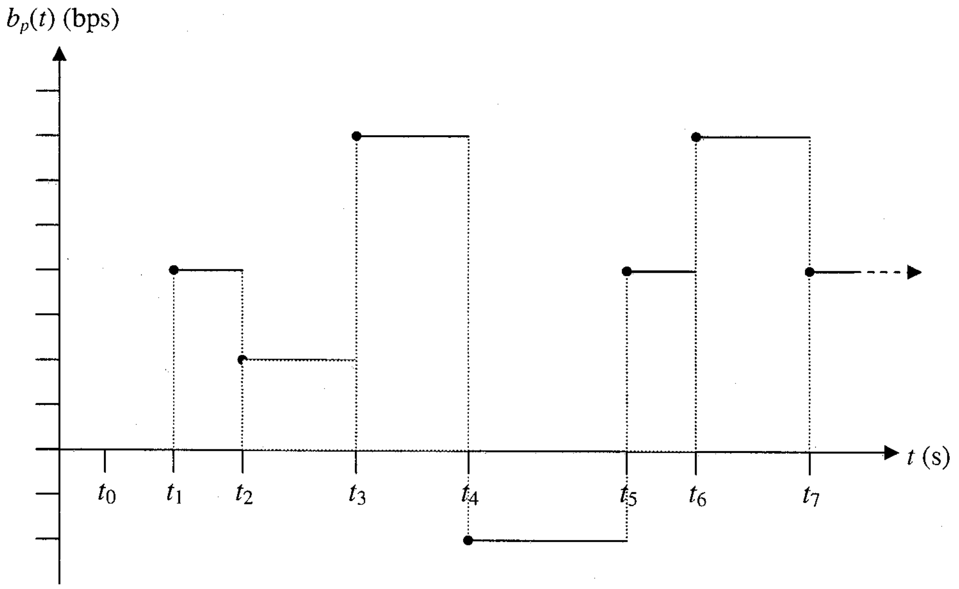

Figure 3-6: Illustration of Net Contribution $C_p(t)$ for a Hypothetical Peer

An operational scenario for a hypothetical peer p that would produce the graph of shown in Figure 3-6 is easily conceived of. Suppose peer p arrives and requests to participate in the streaming overlay at time t_0. At t_1, p is integrated into the overlay, receiving four stripes and forwarding eight copies of one to various children – this results in a net contribution of four units of bandwidth over the interval $[t_1, t_2)$, as shown in Figure 3-6. At time t_2, p's contribution level is reduced by two units, as two of its children leave and no longer need stripes forwarded to them. At t_3, new children arrive and p is instructed to forward five additional copies of a stripe to accommodate them. A negative contribution level is encountered at t_4, when additional departures reduce the number of streams forwarded from p by nine. At t_5, p's net contribution level is increased by six units as it elects to stop receiving two stripes and also starts forwarding copies of a stripe to four new children. Finally, three more copies are forwarded at t_6. Clearly, the grey area in Figure 3-6 represents the integral of the bit-rate $b_p(t)$ over time – thus explaining its interpretation as the net volume of data contributed by p – i.e. $C_p(t)$.

The metric $C_p(t)$ defined above omits at least one piece of information pertinent to any fair characterization of a peer's contribution. In particular, the contribution made by p is strictly a function of its net bit-rate over time and does not in any way account for the relative supply and demand of bandwidth in the streaming session over that period. Thus, a peer contributing to a streaming session where there is an abundance of bandwidth, will be equally rewarded as a peer that contributes the same volume of data to a session where there is a relative scarcity of bandwidth – as an example of this latter scenario, consider a situation in which a flash crowd of resource poor clients produces a sharp, unanswered increase in the demand for bandwidth and leads to the starvation of many participants. To address this concern, $b_p(t)$ is augmented with a "market scaling factor", $\delta(t)$, to arrive at a scaled representation of instantaneous contribution, $s_p(t)$:

$$s_p(t) = \delta(t) \cdot b_p(t)$$

Computing the integral of $s_p(t)$ over time, a new metric, $S_p(t)$ - that characterizes a peer's contribution-level over a period, while also accounting for "market conditions" as described above - is obtained. This metric shall be referred to as the "score" of peer p. A discrete formula for $S_p(t)$ is easily obtained and follows the same pattern as that for $C_p(t)$, with the caveat that $t_0, \ldots, t_n$ now represent the discrete points in time at which $s_p(t)$ may change (i.e. the points at which either $\delta(t)$ or $b_p(t)$ or both may change):

$$S_p(t) = \sum_{i=0}^{n} s_p(t_{i-1})\Delta t_i + s_p(t_n)(t - t_n)$$

With the definitions above, the scaling factor $\delta(t)$ must satisfy a number of important properties in order to obtain scores that provide a meaningful characterization of the relative contributions of participants. First, to address its basic purpose, $\delta(t)$ must positively correlate with the relative demand for bandwidth. More precisely, let $D(t)$ represent the "aggregate demand" for bandwidth at time t – that is, $D(t)$ is the upstream bandwidth capacity required to completely satisfy the needs of all peers interested in receiving a particular stream. Assuming that all peers have sufficient downstream capacity to consume all M stripes into which the stream of interest is divided, then $D(t)$

can be defined as follows, where $n(t)$ is the number of interested peers at time t and again, k is the common bit-rate with which each of the M streams is delivered:

$$D(t) = k \cdot M \cdot n(t)$$

Similarly, let $S(t)$ represent the "aggregate supply" of bandwidth at time t. Thus, if n is the total number of participants registered with the root peer and $f_i^+(t)$ represents the number of stripes that peer i is willing and able to forward at time t, then $S(t)$ can be expressed as:

$$S(t) = \sum_{i=1}^{n} k \cdot f_i^+(t)$$

With the definitions above, we can restate the first desired property of $\delta(t)$ in more concrete terms - $\delta(t)$ should monotonically increase with the ratio $D(t)/S(t)$.

The second desirable property of $\delta(t)$ is that it should be positive. As $s_p(t)$ is defined as the product of $\delta(t)$ and $b_p(t)$, $\delta(t) = 0$ would result in a situation in which a peer making a net contribution of bandwidth to the overlay at time t (i.e. $b_p(t) > 0$) would not increase its score $S_p(t)$ and thus, receive no credit for its cooperative behavior. Similarly, a negative value of $\delta(t)$ would result in a situation where peers contributing to the system would be penalized with lower scores, while those exploiting the overlay (i.e. $b_p(t) < 0$) would be credited.

The third and final requirement of $\delta(t)$ is that it be "normalized" such that valid comparisons can be made between scores accumulated over different sets of streaming sessions. To clarify this point, consider a definition of $\delta(t)$ that monotonically increased with $D(t)$ alone, as opposed to the ratio of $D(t)$ to $S(t)$. In this case, with all other variables held equal, a streaming session with a larger number of clients would have a higher $\delta(t)$ than one with fewer clients and so, a peer p_1 contributing to the larger session would accumulate a higher score than a peer p_2 contributing to the smaller one, regardless of the fact that p_2 may be making a *relative* contribution that is the same or greater than that made by p_1. This bias would preclude a straightforward comparison of the scores of p_1 and p_2 as a valid means for determining which is the superior contributor.

Although many definitions of $\delta(t)$ addressing the requirements above are possible, the trivial formulation below is used in this work.

$$\delta(t) = \frac{D(t)}{S(t)}$$

To conclude this subsection we observe that the notions of score and market scaling factor developed here possess a number of unique advantages when used to quantify peer contributions. First, the accumulative nature of peer scores ensures that clients always have an incentive to make a net contribution to the system, regardless of whether or not they are interested in consuming any stream at a given time, or whether they have only a limited upstream capacity. Second, the market scaling factor maintained by the content provider for each streaming session provides a straightforward means for a potential contributor to identify which session has the greatest relative scarcity of bandwidth. By contributing to this session, which by definition will have the highest value for $\delta(t)$, a peer can optimize the score it may accumulate for a given contribution. In effect, an incentive exists for allocating bandwidth to wherever it is most needed.

Finally, as alluded to earlier, there are no complicated issues of trust to be dealt with – the content provider alone is responsible for maintaining peer scores in a centralized fashion. As the overlay topology is determined by directives issued by the root alone, the score earned or lost by a client at a given time – a function of the number of descriptions consumed and forwarded - is based strictly on the first-hand observations of the content provider itself. To emphasize this key point, it is noted that the root does not depend on peers to obey its directives or truthfully report misbehavior. Should a peer deviate from the root's understanding of the streaming topology by disobeying an order to serve a particular child, the affected child will quickly report this violation, allowing the root to update its picture of the overlay. Conversely, should a peer dishonestly report a misbehaving parent, the root simply finds a new parent for the complainant and frees the falsely accused parent to serve another node – the dishonest peer effectively has no incentive to behave in this manner and save for some loss of opportunity to increase its score, the parent is hardly affected. The use of scores also limits the opportunity for peers to misbehave in some collusive manner - ultimately, the extent to which any set of nodes is able to increase its collective score is a function of the net consumption or contribution of bandwidth made by that set.

3.3.3 Overlay Formation and Maintenance

In Section 3.2 an overview was provided of the manner in which the root peer issues directives to clients - notifying them of their parents and children in the various multicast trees - *after* the root has decided upon the overlay topology to be used at a given time. Here, the scheme by which the root determines this topology is outlined. Note that the algorithm defined here is based on the deterministic tree construction approach employed by CoopNet. Again, however, OmniStream extends this by taking into account the score of each peer and ensuring that clients with higher scores are incorporated into the multicast trees with greater priority than those having lower scores.

As mentioned in Section 3.2, when a client wishes to join a streaming session, it contacts the content provider with a request message. This message contains at least three pieces of information: an indication of the media stream it wishes to receive, the upstream bandwidth capacity that the client is willing and able to contribute to the overlay and finally, the peer's maximum downstream capacity. From these latter two pieces of information, the content provider can determine the maximum in- and out-degree allowed for the node in the topology simply by dividing with the common bit-rate at which each description is to be distributed – in this work, the simplifying assumption is always made that the maximum number of descriptions a peer is willing and able to receive is simply the total number of descriptions, *n*.

Once the join request is received, the content provider immediately increments a running total it maintains for aggregate demand by the downstream capacity of the joining peer – although a similar total is maintained for aggregate supply, this is not incremented until the peer is incorporated into a multicast tree and available to serve children. As in CoopNet, the joining node is then designated to be *fertile* – that is, capable of being an interior node – in one of the *n* multicast trees and *sterile* – limited to being a leaf node - in the rest. One major departure from CoopNet, however, is that a joining peer has the flexibility to declare, in its join request, that it will not contribute any upstream capacity to the overlay – in this case, it is simply designated as sterile in all trees. Regardless, assuming that the joining peer is contributing enough to forward a single description, the determination of which tree the node is to be fertile in is made by keeping track of the number of fertile nodes designated to each tree – the new node is assigned to whichever tree has the fewest fertile members at the time of the join. As noted in [27], the motivation behind allowing a peer to be fertile in at most one tree is rooted in the desire for short trees and disjoint ancestor sets discussed in Section 3.3.1. If contributors are interior nodes in at most one tree each, then their fan-outs are maximized and thus, tree heights are minimized for a given number of nodes. Furthermore, the disjointedness of ancestors is guaranteed in this way – should a client depart or fail, at most one description will be affected.

After determining the fertile layer, if any, the node is added to a *fertile queue* for the designated tree and a *sterile queue* for each of the remaining trees. Note that these queues are new constructs introduced by OmniStream - each of the n multicast trees has its own fertile and sterile queue and these are both priority queues in which precedence is given to nodes with higher scores. Of course, in order to establish this priority, it is assumed that OmniStream's tree management algorithm has access to the latest score of each peer through a peer registry maintained and periodically updated, in a manner to be discussed below, by the content provider.

Having added the newly arrived node to the appropriate queues, the process of updating the topologies of each of the n multicast trees as needed can begin. The same algorithm, outlined here, is separately applied to each tree for this purpose and starts by processing nodes in the fertile queue as follows. First, the peer at the head of the fertile queue, p, is removed and inserted into the associated tree. This insertion occurs by starting at the root of the tree and moving down until a level is found that contains either: (i) a fertile node with a higher score willing and able to accommodate p as a child, (ii) a sterile node or (iii) a fertile node with a lower score. In the event that (i) is encountered, p is added to the topology as the child of the fertile node – should more than one eligible parent exist on the same level, the one with the highest score is selected. In case (ii), the sterile node with the lowest score lower than p is replaced by p and inserted into the appropriate sterile queue. Finally, in case (iii), p is inserted as the parent of the fertile node with the lower score. As mentioned above, once inserted into the tree, the aggregate supply figure maintained by OmniStream is incremented by the upstream capacity that p committed to contribute in its join request.. The process of removing the peer at the head of the fertile queue and inserting it into the tree is repeated until the queue is empty.

Once all nodes in the fertile queue are processed for a tree, those in the sterile queue are handled. The node at the head of the sterile queue, q, is removed and an attempt is made to insert it into the associated multicast tree. This insertion process begins at the root and proceeds down the tree until a level is found that contains either: (i) a fertile node with enough spare upstream bandwidth to accept q as its child, or (ii) a sterile node with a lower score than q. In the event that the former is encountered, q is added as the child of the node with sufficient spare capacity and should more than one suitable parent exist, whichever has the highest score is selected. In case (ii), the sterile node with the lowest score lower than q is bumped from the tree - the tree is updated to replace the sterile node with q and the bumped node is reinserted into the sterile queue. Should the last level of the tree be reached with neither case having been encountered, then the multicast tree has no spare capacity with which to accommodate q and so, the node is simply reinserted into the sterile queue and the insertion attempt is deemed to have failed. The process above, whereby the head of the sterile queue is removed and added to the associated tree, is continued until either the sterile queue is emptied or an insertion attempt fails. In either case, no further updates to the tree are required based on the contents of the sterile queue.

After the algorithm above has completed for each of the multicast trees, OmniStream has effectively computed an up to date overlay topology containing any modifications necessary to accommodate the newly arrived peer. As an aside, note that it is possible for the topology to remain unchanged after a join – for example, a peer that declares that it will contribute no upstream bandwidth will be added as sterile to every tree and if it has a sufficiently low score and none of the trees have the capacity to accommodate it, the new peer will simply sit on the n sterile queues, not having been incorporated into the overlay. Regardless, it is emphasized here that the processing up to this point simply determines what, if any, updates need to be made to overlay and occurs exclusively within the root peer – or whatever content provider host is responsible for running the tree management algorithm; no communication over the network takes place. Only after the updated topology has been computed does the content provider issue directives to the client peers informing them of any new parents and children in each of the multicast trees.

Handling of the other two events that can affect the overlay topology, graceful and ungraceful node departures, also involves the fertile and sterile queues. When a node informs the root of its intention to leave the overlay, it is immediately removed from any queues that may contain it. Further, if the node has been incorporated as a sterile child in any multicast tree, it is detached from its parent and similarly removed. The case in which the departing node has been incorporated into a tree as a fertile child is potentially more complicated – while a fertile node with no children is handled in the same manner as a sterile node, when a fertile node with children is removed, new parents for those children must be found. In this latter scenario, for each tree, we simply add the children orphaned by the departing node to the sterile queue if sterile and to the fertile queue if fertile. Note that a fertile orphan may be the root of a sub-tree itself - only this root is added to the fertile queue and the topology of the orphaned sub-tree itself is unaffected.

Once the queues and trees have been updated as described above in response to the leave request, the same algorithm used to arrive at an equilibrium topology in the case of a node join is applied. For each tree, nodes in the fertile queue are removed in priority sequence and inserted into the tree and subsequently, those in the sterile queue are similarly removed and inserted until the sterile queue is emptied or an insertion attempt fails. Having determined the new topology, the content provider can complete processing of the node leave by appropriately updating its counts of aggregate supply and demand and issuing whatever directives are needed to modify the overlay. As a closing remark on node departures, observe that an ungraceful departure can be processed in the same fashion as a graceful one; in the ungraceful case, however, the processing above is of course only initiated after the delay needed to detect the failed node.

Having covered the handling of node join, leave and failure events, it is important to note that peer dynamics are not the only drivers of change in the overlay topology. As discussed in Section 3.3.2, the scores assigned to peers are continually updated over time as they participate in the streaming infrastructure – should these updates change the relative standing of two peers in an overlay, then a topology update may be needed. For a trivial example of this, consider peers p and q such that $S_q(t_1) < S_p(t_1)$ at some time t_1 and suppose that both peers have elected not to contribute any upstream bandwidth and so, can only join the multicast trees as sterile children. Further, assume that p has been incorporated into all of the available multicast trees, while q waits for service on the corresponding sterile queues. Over time, p, consuming bandwidth and contributing none, will consistently reduce its score while q, neither consuming nor contributing anything, will have its score unchanged. Clearly, at some time t_2 such that $t_2 > t_1$, the condition $S_p(t_2) < S_q(t_2)$ will be satisfied, thus allowing for q to be removed from the sterile queue to replace p in each of the multicast trees, assuming that p is not replaced by some other node first. To accommodate changes of this nature, OmniStream also observes a periodic "update event". More specifically, at regularly scheduled intervals (e.g. once every second), the tree management algorithm updates the scores of all peers in the overlay based on its current topology and then executes the standard algorithm for updating each tree by removing and inserting nodes from its fertile and then its sterile queue. The time between update events is determined by the availability of computational resources – a shorter interval, while allowing for scores to be maintained with greater precision, will obviously place greater demand on the processor(s) available.

Before concluding the outline of OmniStream's tree management algorithm here, two omissions from the discussion above are addressed. First, although it is implied that a simple comparison of peer scores is used when determining whether or not a peer to be inserted into a multicast tree may replace an existing client in that tree, this is not strictly true. Instead, OmniStream divides the peers registered with the streaming infrastructure into N mutually exclusive and approximately equally-sized sets $Q_1, \ldots, Q_N$ – i.e. "quantiles" - based on their scores (e.g. quartiles if $N = 4$, deciles if $N = 10$), such that a peer in a set with a higher index is guaranteed to have a score that is greater than or equal to that of any peer in a set with a lower index. More precisely, if $p \in Q_i$ and $q \in Q_j$ at time t, then $S_p(t) \leq S_q(t)$ if $i < j$. Rather than using raw scores when determining if one peer can bump another, OmniStream actually compares these indices of the sets containing them. The advantage of this approach is the stability it affords the overlay – by avoiding a direct comparison of raw scores in this manner, the frequency with which peers are bumped from multicast trees is reduced without removing the incentive for accumulating higher scores. In fact, with N sets, the number of peers that may be affected by the insertion of a peer is capped at $N - 1$ as a client will only be able to bump a peer in an inferior set. Note that this division of peers is also easily implemented – by maintaining a list of registered peers sorted by score, the index of the set containing a particular client can easily be computed from the position of the peer in the list. In this work, $N = 10$ has been used.

The second and final key point is that, as alluded to in Section 3.3.2, a peer need not be interested in consuming the sub-streams that it forwards. In OmniStream a newly arrived peer may specify in its join request that it is not interested in consuming any stream, but would like to "altruistically" contribute its unused upstream bandwidth – in this case, the tree management algorithm simply incorporates the peer as a fertile node in a single tree, thus optimizing its net contribution to the system. By providing both the means and an incentive – i.e. the opportunity to accumulate a higher score – for peers with idle upstream bandwidth to support the streaming infrastructure, OmniStream, unlike CoopNet, is able to leverage the resources of otherwise uninterested clients.

Another reason for a peer to forward data that it is not consuming itself may be due to an extreme scarcity of bandwidth in a particular streaming session. In this scenario, OmniStream is free to ask a peer with sufficient downstream bandwidth to accept and forward descriptions belonging to the starved session – perhaps while simultaneously consuming descriptions pertaining to whatever other session the forwarding peer is actually interested in. Although this requires the peer to contribute both downstream and upstream bandwidth, the fact that the peer is contributing to a session with scarce bandwidth and therefore a high market scaling factor, gives it an opportunity to quickly increase its score.

CHAPTER 4

Evaluation

A preliminary evaluation of the effectiveness of the OmniStream system has been conducted through the use of simulation experiments. To carry out these experiments, the OmniStream protocol has been implemented in a custom discrete-event system simulator comprised of approximately 2000 lines of Java source code. This simulator is intended to model a period of time over which a swarm of client peers arrive to participate in a single media streaming session with a scheduled start and end time.

The specific simulation parameters used were as follows. Node arrivals, as well as graceful and ungraceful departures were simulated over a 2500 second period. The arrival and planned (i.e. graceful) leave times of the peers were obtained by sampling Gaussian distributions with μ = 350s, σ = 175s and μ = 2150s, σ = 175s, respectively. The intention of clustering joins and leaves within the 2500 second simulation around the times t = 350s and t = 2150s was to capture a scenario analogous to viewers tuning into a particular channel to watch a scheduled 30 minute (2150s – 350s = 1800s = 30min) program and then tuning out. To simulate node failures, a Bernoulli random variable with p = 0.05 was sampled upon the arrival of each node – if the random variable evaluated to true, the node was marked to fail and the time remaining until the failure of the node was fixed by sampling an exponential distribution with 1 / λ = 1800s / 2 = 900s. Otherwise a graceful leave time was determined using the Gaussian distribution described above.

The population participating in the streaming session consisted of some of 400 peers, each of which belonged to one of three different classes. Nodes of Type 1 essentially model free-riders, receiving all available descriptions if possible, but forwarding none. Type 2 nodes model cooperative peers, attempting to receive all available descriptions, but nevertheless reciprocating with a contribution equaling the bandwidth consumed. Finally, Type 3 nodes represent cooperative peers that contribute twice as much bandwidth as they consume. Of the 400 clients in the simulated audience, 10% were of Type 3, 20% of Type 2 and the vast majority (70%) were of Type 1. To capture the fact that these participating end-hosts were the most likely sources of unreliability in the system, each packet was modeled as having a 1% probability of being dropped when moving from one node to the next in the overlay.

Finally, the media stream was divided into eight descriptions, each disseminated using a separate multicast tree. A group of frames (GOF) duration of one second was used and thus, for each second of the media stream, each node could receive at most eight packets, one for each description. Within each of the eight multicast trees, a four second delay was used to model the time needed to detect a failed parent and a delay of one second was used to simulate the service interruptions caused by graceful leaves, or the bumping of nodes by superior clients.

The results obtained using the settings above were promising and are summarized in Table 4-1 and Figure 4-1 below – the data presented is averaged over five runs of the simulation. Table 4-1 summarizes the scores of each class of peers at the end of a 2500 second simulation. Clearly, each class of peers converges on a distinct mean score and this mean increases with the generosity of the associated class. This result demonstrates the viability of the scoring mechanism proposed in this work as a means of quantifying peer contribution-level.

In Figure 4-2, a probability density function (PDF) is provided for each class, characterizing the quality of service experienced by its constituent peers. To construct these graphs the average number of descriptions received by peers of each class is computed for each GOF (i.e. each second) of the simulation – the PDF of these per-second averages can then be plotted as in [27]. Not surprisingly, it is found that nodes of

Type 2 and 3 receive most of the eight descriptions and so should be able to reconstruct the original stream with relatively little distortion. Free-riders, however, are forced to divide whatever surplus bandwidth is provided by Type 2 and 3 nodes amongst themselves and so, on average, receive relatively few descriptions and are heavily penalized with poor streaming quality.

Table 4-1: Node Scores After 2500 Second Simulation

Node Type	*Mean Score*	*Standard Deviation*	*Minimum Score*	*Maximum Score*
Type 1	-6245.94	2.71	-6254.59	-6245.94
Type 2	-10.22	43.11	-57.63	122.15
Type 3	42303.21	2844.47	35366.47	45353.64

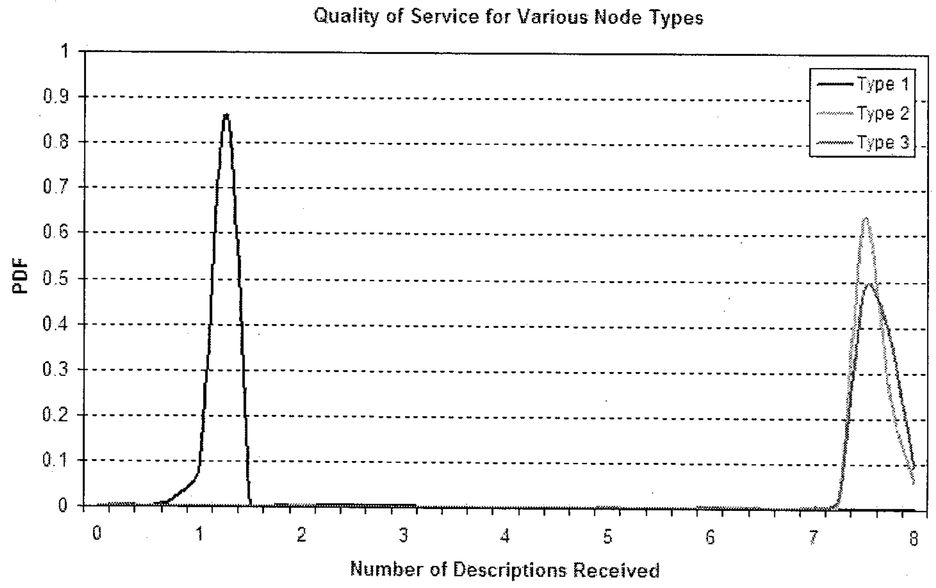

Figure 4-1: PDF of the Average Number of Descriptions Received

CHAPTER 5

Conclusions and Future Work

5.1 Conclusion

Peer-to-peer media streaming represents an attractive means for using the Internet infrastructure to support the live delivery of content to large audiences. In this approach, a server uses basic unicast transmission to forward copies of a media stream to a set of interested client peers. Each of these clients can then make use of their unused upstream bandwidth to forward the stream to additional peers, who can in turn do the same and so on - in theory, this process can recursively span any number of levels from the original source of the stream, thereby sustaining arbitrarily large audiences. By effectively pooling the basic, low-capacity access links of participating peers to support the bandwidth demands of a streaming session, aggregate capacity scales with aggregate demand. In this way, the publisher is spared the undue – and often prohibitive - costs associated with bandwidth provisioning, or the employment of a commercial content distribution network and does not have to depend on sparsely deployed infrastructure, as in the case of IP Multicast.

Unfortunately, this collaborative approach to media distribution is particularly vulnerable to the free-riding behavior all too often exhibited in peer-to-peer systems. Should a substantial proportion of participating clients consume bandwidth without adequately reciprocating by way of their own upstream bandwidth, the scalability of the system will be severely restricted - this is by no means an unlikely scenario, given the self-interested nature of peers under independent administrative control and the substantial utility to be gained by enjoying but not providing the relatively high bandwidth of a media stream.

Several solutions have been developed to address free-riding in peer-to-peer media streaming - these consist of protocols that enforce, or at least provide strong incentives for, cooperative peer behavior. A major limitation of much of this existing work, however, is its strong reliance on an assumption of "tamper-proof" client software, that is, that clients will adhere to the implementation of the prescribed protocol, even though it may be in their individual self-interest to try and circumvent it. This assumption is clearly unrealistic. While alternative approaches exist that do not rely on the obedience of peers in this way, they are affected by other practical issues including slow responsiveness to uncooperative behavior, or an inherently high degree of instability in the streaming overlays they construct.

In this work, OmniStream, a novel extension of the CoopNet peer-to-peer media streaming system, has been developed to address the free-riding problem in this context. In CoopNet, the Multiple Description Coding (MDC) technique is used to divide the stream to be distributed into a set of sub-streams – also referred to as "descriptions", "stripes", or "layers". A client receiving any subset of these descriptions can reconstruct at least part of the original stream, with the level of distortion depending only on the number of descriptions received and of course, decreasing as more are obtained. Alongside MDC, CoopNet also makes use of a centralized tree management algorithm to organize the participating peers into an overlay network comprised of several multicast trees – one for each of the descriptions into which the original stream is divided. The concentration of tree management logic in the server, or "root" peer, is the salient feature of this latter technique. Specifically, the server is responsible for determining the overlay topology and issuing directives to the participating clients as necessary to ensure that they are organized accordingly. In CoopNet, MDC and centralized tree management are applied to achieve redundancy in data and network paths that, in turn, allows for resilience in the face of peer transience. In this work, however, it is recognized that these techniques also provide the basis for a solution to the free-riding problem.

In OmniStream, the root peer takes advantage of centralized tree management to maintain a global view of the overlay topology. With this information, the content provider can determine the number of descriptions received and/or forwarded by each participating client and thus, at any given time, knows the net bandwidth contributed to the overlay by each peer. By computing the integral of this net bandwidth metric over time, the content provider can maintain a score for each peer that accurately characterizes the extent of its contribution to, or exploitation of, the streaming infrastructure over its lifetime. Using MDC, OmniStream can then control the quality of service received by each client based on its score. By ensuring that peers with higher scores are incorporated into the multicast trees used to disseminate descriptions with higher priority – these superior contributors receive a greater fraction of the total number of descriptions and are thus able to reconstruct the original stream with less distortion. This scheme not only provides a strong incentive for peers to contribute upstream bandwidth, so as to accumulate more competitive scores, it also ensures that free-riders, who as net exploiters of the system will always have inferior scores, are never able to consume bandwidth at the expense of cooperative peers.

OmniStream also introduces the concept of a "market scaling factor" that adjusts the net contribution made by a client at a given instant according to the relative demand and supply of bandwidth at that time. This factor positively correlates with the ratio of aggregate demand for bandwidth to its aggregate supply and so, contributions made when there is a relative scarcity of upstream capacity are more greatly valued. Similarly, exploitation under such circumstances is more heavily penalized. The incorporation of peer scores and a market scaling factor represents a significant extension of the CoopNet framework, in that these mechanisms provide a strong impetus for allocative efficiency in the contribution of upstream bandwidth. In simple terms, OmniStream ensures that, at all times, peers have a strong incentive to provide bandwidth wherever it is most needed, regardless of whether or not those peers have any interest in themselves consuming the data they help to distribute.

In this work, a preliminary evaluation of the proposed framework has been conducted through the use of simulation experiments capturing a typical operational scenario. The results obtained clearly demonstrate the effectiveness of peer scores affected by a market factor in characterizing the relative cooperation of clients. The ability of OmniStream to ensure that free-riders are unable to consume bandwidth at the expense of more cooperative participants is also clearly borne out, with free-riders experiencing a substantially reduced quality-of-service over time.

5.2 Future Work

Several directions of research meriting further investigation are immediately evident with the development of OmniStream in this thesis. The points below briefly outline a number of such areas.

(1) The monetization of OmniStream's notion of a peer score is an interesting problem. As OmniStream allows a peer to accumulate greater scores by contributing upstream capacity to streaming sessions over time, defining a scheme whereby a peer can translate this score into some form of payment would ultimately allow providers to purchase bandwidth. In effect, content providers could take advantage of an open, distributed infrastructure in which they are able enlist the upstream bandwidth of other peers on the network, by purchasing it at a rate perhaps related to the market scaling factor for the associated session.

(2) As previously mentioned, a client need not restrict itself to forwarding the particular stream that it is viewing – rather, it is possible for it to forward descriptions belonging to any number of other media streams, regardless of whether or not is has any interest in those streams itself. This "decoupling" is a powerful feature in that it allows a peer to contribute its upstream bandwidth to wherever it may be most effectively employed. Further investigation is needed

to evaluate how successful OmniStream is in optimizing utility through the accommodation of such interaction between streaming sessions.

(3) Although the specifications of the peer score $S_p(t)$ and the market scaling factor $\delta_p(t)$ functions arrived at in this work address the desired requirements of these metrics and produced successful results in the simulation experiments conducted, the space of suitable alternative definitions for $S_p(t)$ and $\delta_p(t)$ is large. Further study of these alternatives may produce specifications that are superior with respect to the ease and efficiency of their implementation, or the results they produce.

(4) As OmniStream tracks the behavior of peers by maintaining scores for each client based on their participation in the streaming overlay, the content provider requires some authentication mechanism that allows it to accurately establish the identity of each peer it communicates with. A suitable authentication mechanism requires some exploration. As an aside, at least one unique requirement of the scheme employed is that some cost be associated with the initial process of acquiring an identity. This is necessary to avoid "whitewashing" behavior, whereby a free-rider sheds a poor reputation – reflected in a low, negative score - simply by acquiring a new identity without any penalty.

(5) In addition to establishing client identity, any practical implementation of OmniStream – or any peer-to-peer media streaming framework for that matter - must also deal with security issues such as ensuring that streaming data is protected from any unauthorized access, or tampering by peers forwarding that data. The means by which to accomplish this must be studied and will likely interrelate with the authentication mechanism employed.

(6) The simulation experiments in this work were driven by hypothetical models of peer composition and dynamics intended to approximate a session in which

clients belonging to only a handful of classes partake in a scheduled broadcast. Clearly, any study aimed at deriving more realistic models of client characteristics and behavior, perhaps through the observation of real peer-to-peer media streaming environments (e.g. [18, 22, 37, 38, 40]), would allow for a more accurate evaluation of OmniStream. Such an empirical investigation may also allow for an accurate utility function characterizing peer behavior to be derived. This would represent a major contribution, in that it would provide an analytical basis with which to reason about the design of related incentive mechanisms.

(7) Although CoopNet is able to sustain loads on the order of tens of thousands of concurrent clients and the associated churn rates of several hundred to a thousand nodes per second using only modest hardware, given the CPU-intensive nature of centralized tree management, as well as the additional computational demands imposed by OmniStream, scalability remains cause for concern. A practical implementation of OmniStream should be developed and experimentally evaluated to more precisely characterize limits on the scalability of this scheme, as well as identify any potential optimizations.

(8) As with CoopNet, OmniStream does not take into account the topology, or other characteristics (e.g. latency), of the underlying physical network when constructing its streaming overlay. Doing so could enable the construction of overlays that (i) minimize delay from the content provider to clients and (ii) employ network resources with greater efficiency by avoiding the duplication of a stream across a physical link. Identifying the manner in which to account for the physical network towards these ends would represent a major enhancement of the protocol.

Bibliography

[1] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[2] Akamai. http://www.akamai.com.

[3] Mayank Bawa, Hrishikesh Deshpande and Hector Garcia-Molina. Transience of Peers and Streaming Media. In *1st Workshop on Hot Topics in Networks*, 2002.

[4] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[5] Yang-Hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas of Communications*, 20(8):1456-1471, 2002.

[6] Yang-Hua Chu and Hui Zhang. Considering Altruism in Peer-to-Peer Internet Streaming Broadcast. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2004.

[7] Yang-Hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan and Hui Zhang. Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *Proceedings of the 2004 USENEX Annual Technical Conference*, 2004.

[8] Yang-Hua Chu, John Chuang and Hui Zhang. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *Proceedings of the 2004 ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 2004.

[9] Yi Cui, Baochun Li and Klara Nahrstedt. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. *IEEE Journal on Selected Areas in Communications*, 22(1):91-106, 2004.

[10] Stephen E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of the 1988 ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 1988.

[11] Hrishikesh Deshpande, Mayank Bawa and Hector Garcia-Molina. Streaming Live Media over Peers. Technical Report 2002-21, Stanford University Database Group, 2002.

[12] Digital Island. http://www.digitalisland.com.

[13] V. K. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 18(5):74-93, 2001.

[14] Ahsan Habib, John Chuang and Mohamed M. Hefeeda. Do We Need Incentive Mechanisms for Peer-to-Peer Media Streaming? In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[15] Ahsan Habib and John Chuang. Incentive Mechanism for Peer-to-Peer Media Streaming. In *Proceedings of the 12th IEEE/ACM International Workshop on Quality of Service*, 2004.

[16] Ahsan Habib and John Chuang. Service Differentiated Peer Selection: An Incentive Mechanism for Peer-to-Peer Media Streaming. *IEEE Transactions on Multimedia*, 8(3):610-621, 2006.

[17] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu and Bharat Bhargava. PROMISE: Peer-to-Peer Media Streaming Using CollectCast. In *Proceedings of the 2003 ACM Multimedia Conference*, 2003.

[18] Yan Huang, Tom Z. J. Fu, Dah-Ming Chiu, John C. S. Lui and Cheng Huang. Challenges, Design and Analysis of a Large-Scale P2P-VoD System. In *Proceedings of the 2008 ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 2008.

[19] Xuxian Jiang, Yu Dong, Dongyan Xu and Bharat Bhargava. GnuStream: A P2P Media Streaming Prototype. In *Proceedings of the 2003 IEEE International Conference on Multimedia and Expo*, 2003.

[20] Justin.tv. http://www.justin.tv.

[21] Sepandar D. Kamvar, Mario T. Schlosser and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th World Wide Web Conference*, 2003.

[22] Karthik Lakshminarayanan and Venkata N. Padmanabhan. Some Findings on the Network Performance of Broadband Hosts. In *Proceedings of the 2003 ACM/USENIX Internet Measurement Conference*, 2003.

[23] William LeFebvre. CNN.com: Facing a World Crisis. Invited talk at the *USENIX Technical Conference*, 2002.

[24] Jin Li. PeerStreaming: A Practical Receiver-Driven Peer-to-Peer Media Streaming System. Technical Report MSR-TR-2004-101, Microsoft Research, 2004.

[25] Wei Tsang Ooi. Dagster: Contributor-Aware End-Host Multicast for Media Streaming in Heterogeneous Environment. In *Proceedings of the 12th Annual SPIE Multimedia Computing and Networking Conference*, 2005.

[26] Tsuen-Wan Ngan, Dan S. Wallach and Peter Druschel. Incentives-Compatible Peer-to-Peer Multicast. In *2nd Workshop on Economics of Peer-to-Peer Systems*, June 2004.

[27] Venkata N. Padmanabhan, Helen J. Wang and Philip A. Chou. Resilient Peer-to-Peer Streaming. In *Proceedings of the 11th IEEE International Conference on Network Protocols*, 2003.

[28] PeerCast. http://www.peercast.org.

[29] Point Topic World Broadband Statistics: Q1 2005. http://www.point-topic.com/dslanalysis/World+Broadband+Statistics+Q1+2005.pdf.

[30] J. A. Pouwlese, J. R. Taal, R. L. Lagendijk, D. H. J. Epema and H. J. Sips. Real-Time Video Delivery Using Peer-to-Peer Bartering Networks and Multiple Description Coding. In *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[31] PPLive. http://www.pplive.com.

[32] Real Networks. http://www.realnetworks.com.

[33] Reza Rejaie and Shad Stafford. A Framework for Architecting Peer-to-Peer Receiver Driven Overlays. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2004.

[34] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[35] Vivek Shrivastava and Suman Banerjee. Natural Selection in Peer-to-Peer Streaming: From the Cathedral to the Bazaar. In *Proceedings of the 15th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2005.

[36] Speedera. http://www.speedera.com.

[37] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs and Hui Zhang. The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points. In *Proceedings of the 2004 ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 2004.

[38] Kunwadee Sripanidkulchai, Bruce Maggs and Hui Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Proceedings of the 2004 ACM/USENIX Internet Measurement Conference*, 2004.

[39] Duc A. Tran, Kien A. Hua and Tai Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[40] Eveline Veloso, Virgilio Almeida, Wagner Meira, Azer Bestavros and Shudong Jin. A Hierachical Characterization of a Live Streaming Media Workload. In *Proceedings of the 2nd ACM/USENIX Internet Measurement Workshop*, 2002.

[41] Stephen B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1995.

[42] Song Ye and Fillia Makedon. Collaboration-Aware Peer-to-Peer Media Streaming. In *Proceedings of the 2004 ACM Multimedia Conference*, 2004.