P^2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment

Tai T. Do, Kien A. Hua, Mounir A. Tantaoui

School of Electrical Engineering

and Computer Science

University of Central Florida

Orlando, FL 32816-2362

Email: {tdo, kienhua, tantaoui}@cs.ucf.edu

Abstract

This paper presents a system for video-on-demand streaming in peer-to-peer environment. We start by realizing the major differences between two types of streaming: live and on-demand. These observations lead to a set of problems that need to be solved for a peer-to-peer video-on-demand system. To address these problems, we propose a solution, which includes detail algorithms for building and maintaining an application multicast tree. The novel ideas in this paper are the use of a new caching scheme at clients, and the introduction of generation for better client management. Performance study based on simulation is carried out. The results show that our system outperforms a recently proposed system [4] in a number of important performance metrics.

I. INTRODUCTION

We are interested in the problem of *Video-on-Demand* (VoD) streaming using peer-to-peer (P2P) approach. Recently, there have been several research projects on *live streaming* using P2P approach [3, 10, 11, 14]. However, applying these techniques into VoD streaming is not a trivial task due to the following

fundamental differences between the two types of streaming. First, end-to-end delay is more important to live streaming than VoD streaming. In live streaming, the shorter the end-to-end delay is, the more lively the stream is perceived by the users (defined as liveness in [3]). In VoD streaming, liveness is simply irrelevant because the video stream is already pre-recorded. This fact implies that while a short tree rooted at the video server and spanned over clients is desirable in live streaming, it is not a necessary condition for the case of VoD streaming. Second, a user joining an on-going live streaming session is only interested in the stream starting from his/her joining time, while in the VoD streaming case the whole video must be delivered to the new user. As such, a good VoD system must find an efficient way to provide the initial missing part of the video to the latecomers. Moreover, the correlations between various variables are different for the two types of streaming. For example, a user will likely stop watching a VoD stream when its QoS degrades, but the user may not do the same thing for a live stream because he/she doesn't have an option of watching it again in the future [6]. Therefore, it is expected that if the QoS of the video stream reduces, many more clients will leave the system in the case of VoD streaming than in the case of live streaming. This observation stretches the importance of a quick and localized failure recovery protocol in a VoD streaming system. It is critical for one to take those differences into account in order to build a successful VoD streaming system in a P2P environment, if the proposed technique is a variant of any existing live streaming systems.

Our proposed technique in this paper is not based on any existing live streaming systems; however, the aforementioned differences are still useful for us to realize challenges in building a P2P VoD streaming system. We consider the following problems as dominant to be solved for a P2P VoD streaming system:

- Quick join: The system must allow a new client to join the system fast. The shorter the joining time is, the better (shorter) the *startup delay* for a client is.
- Fast and localized failure recovery without jitter: Failures are expected to occur more often in a P2P system. The failure recovery protocol should reconnect the abandoned clients smoothly and quickly, so that there are no loss of frame (*jitter*) and no long delay (*glitch*) at client's playback. In addition, the effect of the failure should be localized to ensure that minimal number of clients is affected.

- Effective handling of clients' asynchronous requests: The joining requests of clients arrive to the system at different times. It is expected that the system must deliver the video in full-length to every client without making the server become a bottleneck.
- Small control overhead: Clients in the system may exchange information periodically or non-periodically (on-demand) to keep the system intact. The control overhead must be kept small to make the system scalable.

In this paper, we propose a P2P architecture for VoD streaming, called P^{2}VoD (Peer-to-Peer approach for VoD streaming) to solve the above problems. P^{2}VoD only assumes IP unicast at the network layer. Asynchronous requests of clients are handled by utilizing the clients' resources. Each client in P^{2}VoD has a *variable-size* FIFO buffer to cache the most recent content of the video stream it receives. Existing clients in P^{2}VoD can forward the video stream to a new client as long as they have enough out-bound bandwidth and still hold the first block of the video file in the buffer. We introduce the concept of *generation* and a *novel caching scheme* to deal effectively with failures. The caching scheme allows a group of clients, arriving to the system at different times, to share the same video content in the prefix of their buffers. Such group forms a generation. When a client (or member) in a generation leaves the system, any remaining member of that same generation can provide the video stream without jitter to the abandoned children of the leaving member provided that out-bound bandwidth is sufficient. Control information exchanged between clients is kept small, and most of the time on-demand instead of periodically. As a byproduct , our system also allows new clients to join the system fast by keeping the number of contacts a new client has to make during the join procedure small. We consider the followings as our contributions in this paper:

- Introducing the concept of generation and a novel caching scheme to provide an efficient failure recovery protocol.
- Designing an efficient application multicast tree appropriate for VoD streaming.
- Proposing an efficient control protocol to facilitate the join and failure recovery processes.

There have been attempts in the research community to provide VoD service while taking advantages

of the P2P approach. However, none of these proposals provides a comprehensive or efficient protocol as P^{2}VoD. To the best of our knowledge, Chaining [1] is the first paper applying P2P concept in VoD streaming, but Chaining doesn't provide a recovery protocol in case failures happen. Another proposal is DirectStream [9]. DirectStream handles join and failure recovery by using a directory server. However, this centralized approach presents a single point of failures at the directory server. CoopNet [14] tries to support both live and VoD streaming. In dealing with VoD streaming, CoopNet also has several drawbacks. Clients in CoopNet may need a huge buffer to store the entire video. In case only partial of the video is found at the serving client, the requesting client has to look for the missing part from other clients, which increases the client's startup delay. CoopNet also puts heavy control overhead on the source because the source is required to maintain full knowledge of all distribution trees. In [12], the authors assume many-to-one relationship between supplying-peer and requesting-peer, while we suppose that the relation is one-to-one in our P^{2}VoD system. That paper [12] looks at the problem of media data assignment to minimize the buffer delay, and proposes a technique to amplify the capacity of the streaming system. Perhaps, the closest work to our paper is P2Cast [4]. P2Cast [4] is an extension of Patching, with the exception that a late client can get a patch not only from the server (as in original Patching) but also from other clients. Our P^{2}VoD is different from P2Cast in a couple of ways. First, clients in P^{2}VoD always cache the most recent content of the video stream, while clients in P2Cast only cache the initial part of the video. Due to the caching schemes used, at any time only one stream from an early client is needed to serve a late client in P^{2}VoD, while two such streams, a patching and a base stream, are used at the beginning to deliver the video to a late client in P2Cast. Second, just like SpreadIt [10], P2Cast has to get the source involved whenever a failure occurs, thus vulnerable to disruption due to server bottleneck at the source. In addition, orphaned peers reconnect by using the join algorithm, resulting in long blocking time before their service can resume. On the contrary, in P^{2}VoD, failures are handled locally and most of the times without involvement of the source.

The rest of the paper is organized as follows. Section II presents the architecture and algorithms of the P^{2}VoD system. Performance study based on simulation is presented in section III, in which we compare

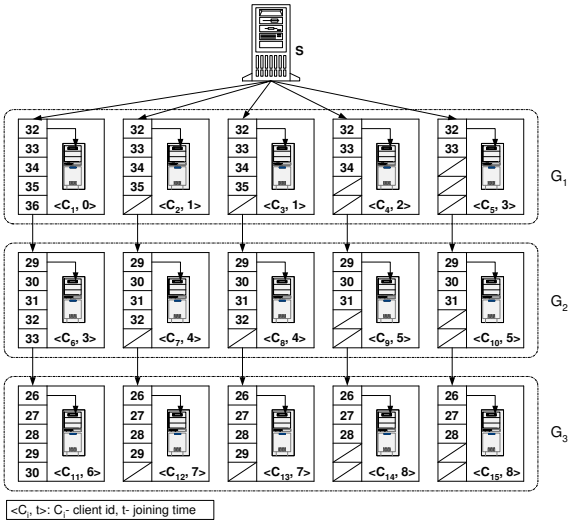

Fig. 1. A snapshot of the P^{2}VoD system at time 36.

P^{2}VoD to a recently proposed system called P2Cast [4]. Finally, we conclude the paper in section IV.

II. Architecture of P^{2}VoD

A. Preliminary

In P^{2}VoD, a streaming connection is assumed to be constant bit-rate, which equals to the playback rate of the video player. We define a retrieval block (R-block) as a data unit of the video, which is also equivalent to one unit of playback time. R-blocks of a video are numbered from 1 to the length of the video file according to their temporal position in the video [1]. Each client has a buffer, whose maximum size is worth of MB units of playback time (i.e, MB R-blocks), to cache the most recent content of the video stream it is receiving. The actual amount of buffer storage used by a client X is denoted as ab_X and lies in the range of $1 \leq ab_X \leq MB$. By using the cache, early arriving client can serve late coming clients by forwarding the stream. If the joining time of a client X is tj_X, then X can serve clients whose joining time is in the range of $[tj_X, tj_X + ab_X]$. Clients having the same smallest numbered R-block in their caches are grouped into a *generation*. Generations are also numbered, starting from G_1 as the oldest generation to G_n as the youngest generation. Clients in these generations, excluding the server, form a *video session*. A video session is closed when none of the clients within that video session still has the first R-block of the video. In this case, a new video session is needed if a new client wants to join the system.

TABLE I
NOTATIONS

Symbol	Meaning
MB	Maximum cache size of a client
ab_X	Actual storage buffer used by a client X
tj_X	Joining time of a client X
add_X	IP address of a client X
$G(X)$	The generation a client X belongs to

Fig. 1 captures a snapshot of a video session of the P^{2}VoD system at time 36. Each client can cache at most 5 R-blocks of the video (MB = 5). However, the actual amount of buffer used by each client is varied. For example, client C_1 uses all available storage of the buffer ($ab_{C_1} = MB = 5$), while client C_4 doesn't ($ab_{C_4} = 3 < MB$). When C_6 arrives to the system at time 3, the first R-block of the video is still in the buffer of C_1. Therefore, C_1 can serve the video stream to C_6. Since C_1, C_2, C_3, C_4 and C_5 all have the same smallest number R-block (R-block 32) in their cache, they form a generation numbered as the first generation G_1.

For the ease of reference, we summarize the notations used throughout the paper in Table I. We also use terms such as peer, client, and user interchangeably in this paper.

B. Data Caching and Generation

Previous schemes for P2P VoD streaming do not handle failure recovery well. Usually when a failure happens, these schemes (e.g. [4]) go back to the server to search for a new parent of the abandoned child. This approach has two serious drawbacks. First, it creates network contention at the server when there is a huge number of clients that need to be reconnected. Second, it increases the reconnection time resulting in unacceptable glitch at client's playback. We overcome these limitations by using a novel caching scheme at the clients and the introduction of generation.

Formally, a generation in P^{2}VoD is a group of clients, who always have the same smallest numbered R-block in their cache. A generation is formed with the help of the caching scheme. Assume X and Y join the same generation at time tj_X and tj_Y, the caching rule is stated as following:

- *Caching Rule*: The difference of actual cache sizes used by two peers X and Y in the same generation

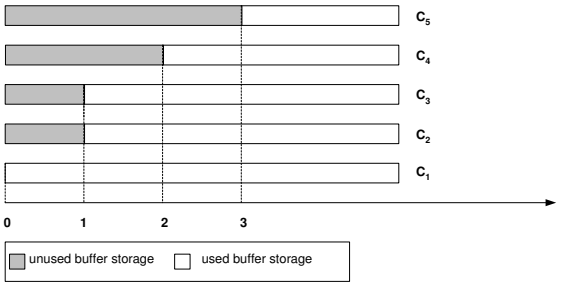

Fig. 2. Caching strategy for clients in the same generation.

must offset the difference in their arrival times, i.e ab_X - ab_Y = tj_Y - tj_X.

Fig. 2 illustrates how peers C_1, C_2, C_3, C_4, and C_5 of generation G_1 from Fig. 1 apply the caching rule. Client C_1 is the first client joining generation G_1 at time 0, and as such can use its maximum available buffer size to cache the video content (ab_{C_1} = MB = 5). At time 1, client C_2 joins generation G_1. To conform to the caching rule, C_2 can't use the maximum available buffer space for caching purpose. As already seen in the Fig. 2, ab_{C_2} = ab_{C_1} - (tj_{C_2} - tj_{C_1}) = 5 - (1- 0) = 4.

The main idea of grouping peers into generations is to provide quick and localized failure recovery without jitter. When a failure happens, peers receiving the video stream from the failing peer are abandoned. In P^{2}VoD, any peer in the same generation with the failing peer can become the substitute to provide the continuing stream without jitter to the abandoned peers, provided that that peer has enough out-bound bandwidth. With the help of the control protocol, which will be discussed later, the list of substituting peers are made available to the abandoned peers in advance, hence ensuring a quick recovery (i.e. minimizing glitch). It is also obvious that the recovery is localized, since it only involves the abandoned peers and peers in the same generation with the failing peer. As an example, consider Fig. 1 and assume that at time 36, peer C_3 fails. As a result, C_8 needs to find a new peer to get the video stream from. P^{2}VoD makes the list of C_1, C_2, C_4, C_5 available to C_8 before the failure to allow a quick and localized recovery for C_8. Since video block 33 is still in the buffer of every peer in the list, C_8 is also guaranteed to have a jitter-free recovery. As a byproduct, generation and caching scheme also help clients to join the system fast, which will be clear later in section II-D.

Even though generations are mainly used for the purpose of failure recovery, they also imply how peers

are connected to each other to get the video stream. The following rules are applied when building generations:

- Generations are indexed by numerical numbers starting from 1. Members of a lower indexed generation arrive to the system no later than any member of other higher indexed generations.
- A peer in generation i-th ($i > 1$) receives the video stream from a peer in generation $(i-1)$-th. Peers at the first generation receive the video stream directly from the server.
- When joining a video session of the system, a new peer either belongs to the current highest numbered generation or be the first member of a newly created generation (indexed higher than the current highest generation) of that video session.

For the remaining of the paper, we use terms such as parent, child, node as same as in conventional tree structure. Two peers are called siblings if they belong to the same generation. The generation where the parent belongs to is called the super generation of the child. A lower indexed generation is called older than a higher indexed generation.

C. Control Protocol

To maintain the connectivity of the system, P^{2}VoD requires peers to exchange the control messages with their neighbors. In P^{2}VoD, neighbors of a peer X include only its parent, children, and siblings.

For its siblings, a peer only needs to keep a list of their IP addresses. We can require peers in the same generation to periodically send *Heart-beat* messages containing the IP addresses to each other in order to keep the list up-to-date. We can also choose to update the list on-demand. That is the update is initiated either by a new node joining the generation or by a node leaving the system intentionally. Of course, by choosing the latter protocol, we have the disadvantage that some of siblings in the list of a peer may no longer exist, since they leave the system involuntarily without triggering the update. The sole purpose of keeping this list is for the failure recovery procedure of peers receiving stream from any peer in $G(X)$. Therefore, out-of-date list of siblings may cause longer recovery time, but will not make the failure recovery algorithm run incorrectly.

For its child, when agreeing to forward the video stream, X only needs to send the IP addresses of X and its siblings to its child. This message is sent only once if X exchanges the control message with its siblings on-demand. Otherwise, periodic refresh is also necessary if X and its siblings communicate periodically.

X also sends its IP address to its parent when joining the system.

In addition, we require X to send the following control information to the server when joining the system: $<add_X, G(X), t^X_{exp}>$, where t^X_{exp} is the expiration time when the first R-block of the video is no longer in the cache of X. This message exchange is used to facilitate the join process in section II-D. We argue that since this control information is sent to the server only once during the lifetime in the system of a peer, it shouldn't pose any burden on the server.

D. Join Algorithm

Due to the message exchange in the control protocol, server S has the list of peers at the youngest generation for each video session, denoted as G_y, and their corresponding $t^{G_y}_{exp}$. Note that the expiration time, $t^{G_y}_{exp}$, is the same for every member of the youngest generation. Initially, when the system is empty, $t^{G_y}_{exp}$ is set to minus infinity and the set of youngest generation peers contains only S.

Case 1: If all of the existing video sessions are closed, X will be admitted if server S still has enough outbound-bandwidth; otherwise, X will be rejected. In the former case, a new video session is created, and X is the first member of the first generation of that video session.

Case 2: For the case where there is at least one existing video session still open, X will try to join that video session. X first contacts the server to obtain the list of peers in the current youngest generation of that open video session, and the join algorithm proceeds as follows.

- *Step 1*: X contacts a random member of the G_y set, and acquires the list of peers at the super generation of G_y.
- *Step 2*: If t_{exp} at the super generation is greater than tj_X, then go to Step 3, otherwise go to Step 4.
- *Step 3*: X selects a peer from the super generation to become its parent according to some *parent selection criteria*, which will be discussed shortly. If such a candidate exists, X is accepted into the

system. X becomes a member of G_y and starts receiving the video stream from its selected parent. ab_X is set to conform to the caching rule.

- *Step 4*: A new generation is formed below G_y in the hierarchy, and X is made the first member of that generation. X uses the same parent selection criteria as used in Step 3 to select a peer from G_y as its parent, and ab_X is set to equal to MB. If such a candidate exists, X is accepted into the system. If no such peer exists, X will try other open video sessions if existed. Otherwise, X last tries to join the system as in Case 1.

We also introduce a new parameter K to limit the maximum number of clients allowed in the first generation of each video session. This helps to avoid too many direct connections established at the server in case there is a large burst of clients arriving to the system together at the early phase of an open video session.

Parent selection algrothims are heuristic solutions, trying to match the application overlay with the underlying network more efficiently. Further discussion can be found in [19]. We propose three different parent selection algorithms:

- *Round Robin Selection*: This selection promotes the fairness among peers. Among peers who have enough out-bound bandwidth, the requesting peer should select a peer, who hasn't served any client for the longest period of time.
- *Smallest Delay Selection*: This strategy aims to minimize the joining time, hence startup delay, of the requesting peer. The requesting peer picks the first peer in the candidate group, who has enough out-bound bandwidth.
- *Smallest Distance Selection*: The goal of this selection is to reduce the number of links involved in the underlying network. The requesting peer chooses a peer, who has enough out-bound bandwidth and closest to it.

E. Failure Recovery

In a peer-to-peer environment like P^{2}VoD, failures are expected to happen often due to the unpredictable behavior of users as well as the congested traffic in the underlying network. P^{2}VoD uses a two-phase failure

recovery protocol. The first phase requires a peer to detect a failure between it and the parent. Then the abandoned peer enters the second phase to recover from the failure by finding a new parent. We discuss these phases in the following subsections.

1) Detecting failures: Failures in P^{2}VoD are categorized into two kinds: grateful and unexpected. Grateful failure refers to the case when a peer intentionally chooses to leave the system. On the other hand, unexpected failure accounts for network failure and client crash (e.x, buffer overflow). These two kinds of failures require different detection mechanisms. Again, due to the length constraint of the paper, details of the detection machanisms are presented in [19].

2) Recovering from failures: When there is a failure at a peer X, the whole sub-tree under X is affected. X initiates the recovery process, while the rest of the sub-tree are informed to wait through the use of the WAIT message. If X succeeds, the whole sub-tree is recovered. If X fails to find a new parent, then X is rejected. The immediate children of X will invoke the recovery process to try to recover their own sub-tree. The process repeats recursively through the whole sub-tree. A disrupted peer X uses the following steps to find a new parent for itself.

- *Step 1*: X contacts the siblings of its parent to find a new parent according to one of the three parent selection criteria. If there exists such parent, X recovers, otherwise go to Step 2.
- *Step 2*: X contacts the server. X is recovered and connected to S if the server still has enough out-bound bandwidth. Otherwise, X is rejected.

III. Performance Evaluation

We use simulation method to evaluate the performance of P^{2}VoD. Four performace metrics are used: (1) *Rejection probability*- the probability that a client tries to join the system but can not get the service. This metric illustrates which protocol uses the network resources (bandwidth in this case) more effectively; (2) *Number of contacts during the join*- the number of nodes a joining client has to contact before it can start getting the service. It is an indirect measurement of the startup delay for a client in the system; (3) *Server stress*- the amount of bandwith used at the server, or the number of streams established at the server. It is

an idicator of how congested (on average) the video server is; (4) *Number of contacts during the recovery*- the number of nodes a client has to contact during the recovery process. It indicates how fast a client can recover from a failure.

We study different scenarios using the aforementioned metrics. Firstly, we examine the effects of two adjustable parameters in P^{2}VoD: the maximum number of clients allowed in the first generation of each video session, and the maximum buffer size each client can use, denoted as K and MB. Secondly, we compare P^{2}VoD with P2Cast [4]. Since the simulation results on different parent selection algorithms (section II-D) are similar, the results reported in this paper for P^{2}VoD are based only on the Smallest Delay parent selection algorithm.

In our simulation, the underlying network topology is created using GT-ITM [18]. The whole network consists of one transit network (with 4 nodes), and 12 stub domains (with 96 nodes in total). We assume that each node in this network represents a local area network with the ability to host unlimited number of clients, and to have enough bandwidth to support media streaming. Routing between two nodes is determined by using the shortest path algorithm. We assign bandwidth capacity to links in the network as follows: a backbone link (at least one end point of the link is a transit node) can support 25 concurrent media streams, and an edge link (any link in the network, which is not a backbone link) can support 7 concurrent media streams.

The video server is placed at a transit node. Clients arriving to the system follow the Poisson distribution, and are randomly placed into one of the network nodes. There is only one video available at the server with length of two hours. Each simulation run also lasts for two hours. The total number of clients arriving to the system during a simulation run is determined as (Arrival Rate * Length of the Simulation Run), called normlized workload.

A. Scenario 1: Effects of the adjustable parameters

In this scenario, we assume that there is no failure. We study the effects of the two adjustable parameters in the protocol, K and MB, with regard to two performance metrics: the rejection probability, and the server

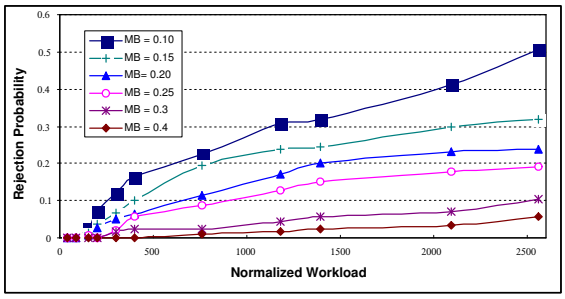

Fig. 3. Rejection probability for various maxium buffer sizes (K = 3).

stress. The two parameters are examined separately, because they are orthogonal to each other.

We first look at the effects of MB by fixing the other parameter $K = 3$. Fig. 3 shows the rejection probability as a function of MB and the normalized workload. The value of MB is varied from 0.1 to 0.4 of the length of the video. With a fix value of MB, the rejection probability increases with the increasing size of the normalized workload. This is due to the limited available bandwidth of the underlying network. When there are already a large number of clients in the system, the probability that a new client can find a path with sufficient bandwidth to get the stream from any existing client is smaller. A more interesting result is that the rejection probability can be substantially reduced by increasing the size of the MB. When doubling the size of MB (from 0.1 to 0.2, and 0.2 to 0.4), the rejection probability is reduced by half. Since the rejection probability is a good indicator of how scalable a system is, MB can be used to adjust the system to a desirable scalability.

We also investigate the correlation between MB and the server stress. What we find out is that they are not directly correlated to each other, so we opt not to report that result in this paper. For the readers information, the maximum number of streams established at the server never exceeds 12.

Fig. 4 illustrates the relationship between server stress and parameter K when fixing the value of MB at 0.1. The simulation is also tested on various normalized workloads as in the previous scenario for MB. The values of K are selectively picked in the range from 2 to 10. Our first observation is that the server is most stressed when the workload is lightest. This seemingly counter-intuitive phenomemon can be explained as follows. When the workload is light, clients arriving to the system are far away from each other in time. Therefore, a new client hardly finds an open video session to join, which implies that the server has to create

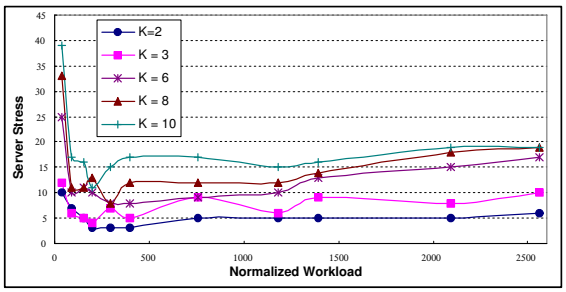

Fig. 4. Server stress with various values of K ($MB = 0.1$).

a new video session for almost every client. On the other hand, when the workload on the system is heavier, clients arriving to the system are close to each other in time. This makes a new client be more likely to find an open video session to join, which also means a new client usually gets the video stream from another client, not the server. In short, almost every clients get the video stream directly from the server in the case of light workload, while in the case of heavy workload most clients get the video stream from some other client, not the sever. This property of P^{2}VoD is extremely important for system scalability, especially when the available bandwidth at the server is limited. The second result derived from Fig. 4 is the server is less congested when K is small. However, the difference in server stress for various values of K is not significant. Besides, having a relatively large K makes the video session more stable when failures happen, i.e with small K a failure at the first generation will likely cause disruption to the whole video session, while with larger K such failure only causes a portion of the video session to be in recovery mode. Hence, the recommendation is to use a large K (e.x 8 or 10).

Similar to the case of MB, we also carry out experiments to find the relationship between K and the rejection probability. It turns out that different values of K give us a very similar curve, much like the one in Fig. 3 when $K = 3$ and $MB = 0.1$.

B. Scenario 2: P^{2}VoD vs. P2Cast

In this scenario, we compare the performances of our protocol (P^{2}VoD with Smallest Delay, hereafter we only use P^{2}VoD for short) against two variants of P2Cast [4]: the BF-delay-appro and BF protocols. These two variants of P2Cast are chosen because, as reported in [4], BF has the best performance in server stress,

while BF-delay-appro achieves lowest rejection probability. For the P^{2}VoD protocol, we use K = 6 and MB = 0.2. The threshold for P2Cast protocols is also set at 0.2. In other words, in our simulation P^{2}VoD and P2Cast require the same amount of buffer storage for each client.

First, we simulate the protocols with varying workloads and without failures. As shown in Fig. 5, P^{2}VoD outperforms BF-delay-appro and BF in term of rejection probability, especially when the workload increases. One possible explanation is that P2Cast variants require each client to obtain two streams, a patch and a base stream, when joining the system, while P^{2}VoD only needs to allocate one single stream for a new coming client. When the workload increases (i.e the arrival rate increases), a significant amount of network bandwidth allocated for patch streams in P2Cast system are not released when new clients join, resulting in higher rejection probability. As shown in Fig. 6, P^{2}VoD is also less demanding on the server in term of network bandwidth comparing to the two variants of P2Cast. In P^{2}VoD, a video session is open as long as clients in the youngest generation still have the first block of the video in their buffer. Meanwhile in P2Cast system, a video session is only open for a short period of time starting when the first client join the video session and last for the length of the threshold. The result is there are more video sessions in P2Cast than in P^{2}VoD, leading to heavier bandwidth requirement to the server in P2Cast than in P^{2}VoD. This advantage of P^{2}VoD is most useful when the server has limited bandwidth. Even though we do not carry out the simulation in case the server is placed at a stub domain (i.e the server has less bandwidth than when it is placed in a transit domain), we expect that our protocol will outperform P2Cast more significantly regarding of rejection probability and server stress. Fig. 7 illustrates that when joining the system, on average a new client in P^{2}VoD has to contact far less number of existing clients than in P2Cast. Between two variants of P2Cast, the reason that a new client in BF-delay-appro contacts less number of existing clients than in BF is obvious. In BF-delay-appro, a new client only needs to find an existing client with enough bandwidth to support the stream, while in BF a new client has to look for the ”fattest pipe” [4] to get the stream from. In spite of that, our P^{2}VoD still performs better than BF-delay-appro due to several reasons. Firstly, a new client in P^{2}VoD only needs to contact existing clients in the youngest generation and the super generation

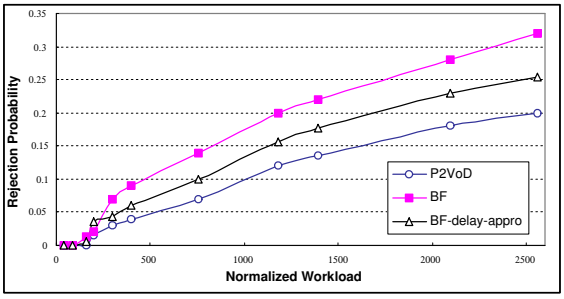

Fig. 5. P^{2}VoD vs. P2Cast: Client rejection probability.

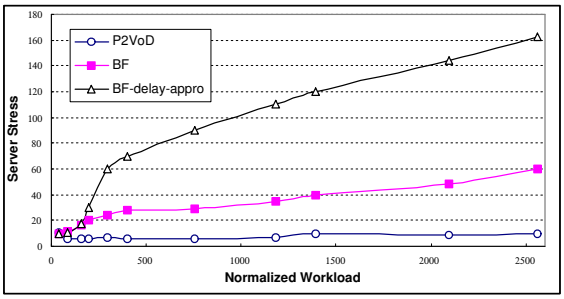

Fig. 6. P^{2}VoD vs. P2Cast: Server Stress.

of that, while a new client in BF-delay-appro may have to contact every existing clients in the video session it wants to join. Secondly, a new client in P^{2}VoD only looks for one existing client to get the stream from, while a new client in BF-delay-appro needs to look for two existing clients to get two separate streams, the patch and the base stream.

To simulate client failures, we first allow 3516 clients to join each system, which is equivalent to having clients arrive to the systems at a rate of 0.4 per second during a period of 2 hours. After the joining process is finished, we assign a failure probability to each client, and then observe how affected clients recover from failures. Note that we simulate the protocols in an environment where failures do not happen very often

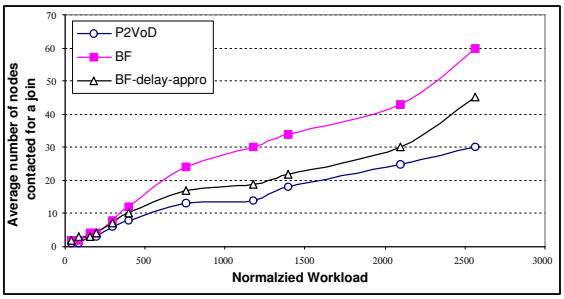

Fig. 7. P^{2}VoD vs. P2Cast: Join overhead.

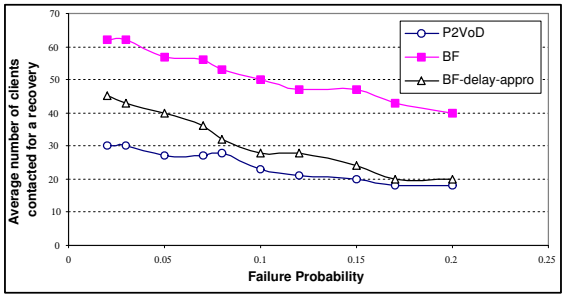

Fig. 8. P^2VoD vs. P2Cast: Failure overhead.

(failure probability goes from 0.05 to 0.2). Fig. 8 shows that the number of clients an affected client has to contact during a recovery in P2Cast only equals to one half of that in BF and two third of that in BF-delay-appro. When the failure probability moderately increases (from 0.05 to 0.2), the number of clients contacted during a recovery decreases. This is perfectly normal because when more clients leave the system, network bandwidth is also released; hence, this makes easier for an affected client to find a new parent to get the stream from.

IV. Conclusion

In this paper, we have presented a new system for Video-On-Demand streaming in a Peer-to-Peer environment. The novel ideas in this paper are the introduction of *generation* and *caching scheme*. Together with an efficient control protocol, they help P^2VoD to allow clients join the system fast, as well as to recover failures in a quick and localized manner. Simulation-based performance study shows that our protocol is superior than a recently proposed system named P2Cast [4]. With the same underlying network and workload, P^2VoD allows more clients to join the system than variants of P2Cast, yet at the same time network demand on the server in P^2VoD is also significantly less than in P2Cast. Simulation results also reveal that P^2VoD allows clients join the system faster and also requires less time for an affected client to recover from a failure than P2Cast.

Acknowledgment

The authors would like to thank the US National Science Foundation for partially funding the work in this paper under grant ANI-0088026.

REFERENCES

[1] S. Sheu, K. Hua, W. Tavanapong, "Chaining: A Generalized Batching Technique for Video-on-Demand Systems," in *IEEE ICMCS 1997*.

[2] D. Tran, K. Hua, S. Sheu, "A New Caching Architecture for Efficient Video-on-Deman Services on the Internet," in *IEEE SAINT 2003*.

[3] D. Tran, K. Hua, T. Do, "A Peer-to-Peer Architecture for Media Streaming," in *IEEE JSAC 2003*.

[4] Y. Guo, K. Suh, J. Kurose, D. Towsley, "P2Cast: P2P Patching Scheme for VoD Service," in *WWW 2003*.

[5] S. Saroiu, P. Gummadi, S.Gribble, "A measurement study of peer-to-peer file sharing systems," in *ACM/SPIE MMCN 2002*.

[6] E. Velos, V. Almeida, W. Meira, A. Bestavros, S. Jin, "A Hierarchical Characterization of a Live Streaming Media Workload," in *IEEE IMW 2002*.

[7] K. Hua, M. Tantaoui, "Cost Effective and Scalable Video Streaming Techniques," in *Borko Furht, Oge Marques (eds.): Handbook of Video Databases 2002*.

[8] Y. Chu, S. Rao, H. Zang, "A Case for End System Multicast," in *ACM SIGMETRICS 2000*.

[9] Y. Guo, K. Suh, J. Kurose, D. Towsley, "A Peer-to-Peer On-Demand Streaming Service and Its Performance Evaluation," in *IEEE ICME 2003*.

[10] H. Deshpande, M. Bawa, H. Garcia-Molina, "Streaming live media over a peer-to-peer network," in *Work at CS-Stanford 2002*.

[11] S. Banerjee, B. Bhattacharjee, C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM 2002*.

[12] D. XU, M. Hefeeda, S. Hambrusch, B. Bhargava, "On peer-to-peer media streaming," in *IEEE ICDCS 2002*.

[13] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proc. of the Fourth Symposium on Operating Systems Design and Implementation 2002*.

[14] V. Padmanabhan, H. Wang, P. Chou, "Distributing Streaming Media Content Using Cooperative Networking," in *NOSSDAV 2002*.

[15] M. Bawa, H. Deshpande, H. Garica-Molina, "Transience of Peers and Streaming Media," in *ACM SIGCOMM Computer Commuications Review 2003*.

[16] S. Q Zhuang, B. Y. Zhao, D. D. Joseph, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *ACM/IEEE NOSSDAV 2001*.

[17] D. Pendakaris, S. Shi, "ALMI: An application level multicast infrastructure," in *USENIX Symposium on Internet Tchnologies and Systems 2001*.

[18] E. W. Zegura, K. Calvert, S.Bhattacharjee, "How to model an Internetwork," in *IEEE Infocom 1996*.

[19] T. Do, K. Hua, M. Tantaoui, "P^2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," in *tech. rep. 2003*, SEECS, UCF, http://www.cs.ucf.edu/~tdo/