NAP: An Agent-based Scheme on Reducing Churn-Induced Delays for P2P Live Streaming

Abstract

Peer-to-peer (P2P) multimedia streaming provides a scalable solution for IPTV. However, delays from channel switch and streaming recovery are typically in the scale of 10-60 seconds, which have hindered the extensive commercial deployment of P2P systems. We call those two types of delays, churn-induced delays. Obtaining assurances on churn-induced delays in dynamic and heterogeneous network environments is a challenge. In this paper, we devise a simple, yet efficient agent-based P2P streaming scheme, called NAP, which reduces churn-induced delays. We first formulate the problems of minimizing channel-switching delay and streaming recovery delay. We then present the detailed methodology of NAP. In addition, we develop a queuing model for the P2P streaming scenario and analyze the properties of NAP based on that model. Our numerical study reveals the effectiveness of NAP, and shows that it significantly reduces churn-induced delays, especially channel-switching delays.

1. Introduction

By way of efficient cooperation among end users, P2P networks dramatically increase scalability, and thus exhibit a growing popularity in the application domain of multimedia streaming. As a result, many P2P live video systems, such as PPLive, Coolstreaming, and Sopcast, have been successfully deployed in the recent years, and most of them have over 100 channels, prevailing with millions of users [1], [2]. However, recent measurement studies of P2P streaming indicate that the detrimental impacts from node churn, long switching delay, and playback lag are hindering the extensive commercial deployment of P2P systems [3]. For example, IPTV deployment volume from commercial service providers is far below the industry expectation [4].

Besides playback lag, delays occurring in P2P streaming may arise from two factors: node churn and channel switching. Node churn represents the frequent event of peer arrival or departure. Typically, peer departure can lead to streaming interruption to the downstream peers, where delay happens during the restoration of streaming connections [3]. We call such delay *recovery delay*. Another type of churn is channel churn, which represents the event of channel switching in a multi-channel P2P streaming system. It has been observed that channel churn is much more frequent than node churn [5], and can cause severe instability to the system. In a multi-channel P2P streaming system, the video content of each channel is distributed by a different swarm of peers. Similar to node churn, when a peer switches from channel A to channel B, its downstream peers need to locate new data feed from other peers in the swarm of A. Additionally, channel switching delay occurs as well when this peer attempts new stream connections in the swarm of B. As we can see, these types of delays stem from the churns of node departure and channel switching. We call such delays *churn-induced delays*. Currently, users have been accustomed to delays under seconds, which are typical in a cable TV system [5]. However, churn-induced delays are significant in current P2P systems. For example, measurement studies indicate that channel switching delays are typically in the scale of 10-60 seconds [6]. From the perspective of user experience, this is obviously undesirable. Motivated by this, we propose *NAP*, a Novel, Agent-based P2P scheme to reduce the churn-induced delay. For minimizing playback lag, there exists a vast array of solutions, such as [7], [8]. NAP can adopt any of these minimum-playback-delay algorithms to yield low-delay streaming.

In this paper, we focus on reducing the churn-induced delays, including recovery delay and channel switching delay. We develop NAP, an agent based solution. Based on the fact that churn-induced delays identically stem from the time of re-connecting to new peers, NAP is devised with preventive connections to all channels. Once an actual connection is requested, time is saved in retrieving bootstrapping information, and obtaining authorization and authentication. However, maintaining preventive connections for all peers to all channels makes it impractical due to significant number of control signals and message overhead.Towards an efficient control and reduced message overhead, we propose that for each channel some powerful peers are selected as agents to represent the peers in the channel. The agents distill the bootstrapping peers with superior bandwidth and lifetime expectation to quickly serve the viewer in the initial period of streaming. Moreover, the agents pre-schedule the downloading plan about data chunk partition and streaming for future viewers, and coordinate with each other to build the preventive connections for peers represented by them. Since agents work in a distributed manner whose amount fluctuates

adaptively to the network size, NAP exhibits massive scalability. Due to space constraint, we will not discuss security concerns in this paper. As we know, NAP can adopt existing security schemes, which has been extensively studied in previous works [9].

This paper focuses on reducing the churn-induced delays and presents the theoretical bound on the efficiency of NAP. Although similar schemes, called proxy methods, were studied in [10], [11], previous research mainly focuses on reliability and scalability, which does not work towards delay optimization. In summary, the paper makes three important contributions: (1) we develop NAP, an efficient agent-based scheme towards reducing the churn-induced delay in P2P streaming with reasonable message overheads; (2) we develop a queuing theory model for NAP based on P2P streaming scenario, which can also be generally applied to other preemptive schemes after simple modification. By virtue of this model, we analyze the theoretical properties of NAP; and (3) we numerically analyze the performance of NAP and show the improved delay properties of NAP based on both queuing theory model and simulation results. Our experimental results indicate that NAP can significantly reduce churn-induced delays, especially the channel-switching delay.

The rest of this paper is organized as follows. Section 2 describes an overview of past and related works. In Section 3, we formulate the problems of reducing churn-induced delays, and present our agent-based solution – NAP. In Section 4, we model the NAP and justify its performance by the queuing theory model we develop for P2P scenario specifically. The results of numerical studies in Section 5 validate the effective performance of NAP at the end of this chapter. Section 6 concludes the paper.

2. Related Work

Previous works on IPTV show that the next channel that a user may watch can be predicted using probability analysis based on user's past channel switching patterns [12]–[14]. For example, a viewer watching a live news channel may switch to another news channel with high probability. Also, it is highly probable that a viewer may switch to an adjacent channel in the channel list when he or she is seeking some interesting channel in sequence. Thus, several papers such as [12], [13], on multicast IPTV systems propose to send contents of the predicted next channels in parallel with the currently viewed channel. Should the user switch to one of the predicted channels, he or she can immediately watch the next channel without delay. However, such a method is bandwidth consuming in transmitting multiple streams, which is not practical for current P2P streaming systems due to limited upload and download bandwidth.

The problem of reducing churn-induced delay in P2P streaming has close relationship to the problem of improving the churn resilience, because a resilient system generally has less streaming interruptions. In [5], Wu propose a strategy, called view-upload decoupling (VUD), to reduce the channel churn. The VUD-based P2P streaming approach decouples peers' viewing from their content uploading. In other words, what a peer uploads is independent of what it views [3]. The major advantages of VUD-based systems are inherent robustness in the high-channel-churn environments and flexible cross-channel resource sharing capability. The drawback of this approach is that VUD costs more bandwidth overhead. To tackle this problem, Wu *et al.* [3] propose a substream-swarming enhancement. Specifically, each swarm only distributes a small portion of the stream, called a substream, to viewers. The viewers need to collect substreams from multiple swarms for a complete streaming. This substream concept can improve the viewing performance without significant bandwidth overhead. As we can see, in VUD, the distribution swarms are more stable than traditional systems, where peers deliver the same channel content that they are viewing. As a churn-resilient scheme, the channel-switching behavior of a peer in VUD will not influence its downstream peers. Yet, VUD can neither reduce the delay consumed at the peer that is switching the channel, nor the delay from the node churn.

Instead of mitigating one type of delay, our strategy focuses on retrenching the connection time for all cases of churn-induced delays. Additionally, considering the promising advantages of VUD, NAP is capable of integrating our scheme with VUD to obtain better performance.

3. NAP: An Agent-Based P2P Scheme

For churns in P2P live streaming systems, we can generally classify them into two categories: *peer churn* and *channel churn*. Peer churn arises from peer arrival and departure [1], and channel churn comes from channel switching [3]. In this section, we first formulate the problems of reducing delays caused by the two types of churns: node churn and channel churn, and then we describe the methodology of NAP to reduce these delays.

3.1. On Reducing Delay from Channel Churn

We consider a peer-to-peer streaming system with Q channels, where each channel is associated with a distinctive streaming overlay. We denote the set of channels as $\mathbf{C}$. Given a channel $C_i \in \mathbf{C}$, its streaming overlay can be modeled as $G_i = (V_i, E_i)$, where V_i is the set of vertices representing peer nodes on this overlay, and E_i is the set of overlay edges representing directed overlay streaming links. Each channel $C_i \in \mathbf{C}$ is considered as a streaming session, originating from a single source node S_i to a set of receivers R_i, where $V_i = \{S_i\} \cup R_i$. Suppose S_i streams data at a constant streaming rate of s_i units/second. If a peer p viewing channel

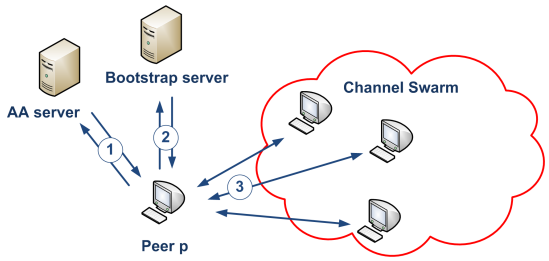


Figure 1: General Scheme: (1) *Authentication and authorization*; (2) *Peer list download*; (3) *Contact other peers, retrieve chunk map and more lists of connection candidates, and then schedule downloading plan.*

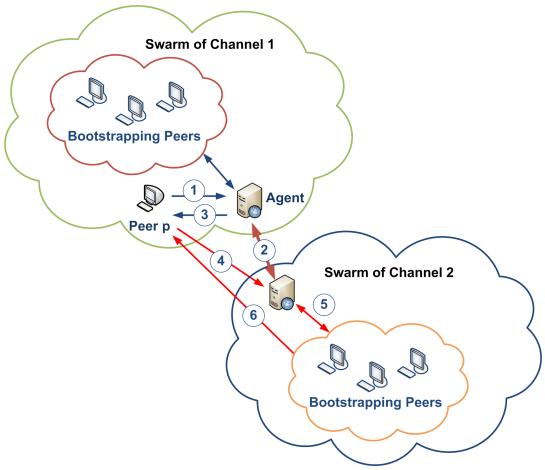


Figure 2: NAP: (1) *Peer registers at the agent*; (2) *Agents periodically exchange information with each other, distill bootstrapping peers and pre-schedule the downloading plan based on chunk information on BP*; (3) *Peer downloads peer list and list of agents in other channels*; (4) *When switches channel, peer sends request to agent in that channel and also contacts ordinary peers in the list simultaneously*; (5) *Agent forwards request to bootstrapping peers*; (6) *Bootstrapping peers directly send data to peer.*

C_i receives the aggregated stream at s_i units/second from its parents, we call peer p as *fully served* [7]. We assume that a fully served peer can smoothly play back the streaming content at its original rate of s_i units/second [7].

To understand the switching delay problem, we first describe the general channel switching scheme without switching delay optimization [15]. As illustrated in Figure 1, when switching to a new channel C_i, a peer p initiates a connection with an *AA server* for Authentication and Authorization [15]. Afterwards, it contacts with a *bootstrap server* (BS), which is a server maintaining a partial list L of current peers on the overlay of C_i. BS sends p a list L_R of random peers as the connection candidates for p, where $L_R \subset L$ [15]. To distinguish from the bootstrapping peers which is mentioned later in NAP, we call here the list of peers retrieved from BS as *ordinary peers*. Then p requests streaming from peers in L_R. Active peers in L_R response with their IP addresses, data chunk information and additional lists of connection candidates in the swarm. If the cumulative available bandwidth from responding peers in L_R satisfies the streaming rate and the cumulative chunk is complete, p schedules downloading plan and attempts the streaming process with them.

If the above bootstrap procedure is successful, the overlay mesh for channel C_i is updated, and subsequently p receives the channel content from its upstream peers. Clearly, the expected channel switching delay for peer p, denoted as $\overline{d}_{sw}$, can be expressed by

$$\overline{d}_{sw} = \overline{d}_A + \overline{d}_{ls} + \overline{d}_p + \overline{d}_{sc}, \quad (1)$$

where $\overline{d}_A$ is the expected time cost in authorization and authentication before watching a channel, $\overline{d}_{ls}$ is the expected delay to retrieve a peer list of the target channel from the bootstrap server, $\overline{d}_p$ is the expected time spent in initializing the contact with those peers in the list, and $\overline{d}_{sc}$ includes the delay in exchanging data chunk information with responding peers in the list, the time to schedule downloading different chunks from specific peers and the communication delay to receive the chunk data. We now define the problem formally:

Definition 1: **Minimum Channel Switching Delay Problem (MCSD problem):** Given the average time required to complete each step in channel switching process, i.e., $\overline{d}_A, \overline{d}_{ls}, \overline{d}_p$, and $\overline{d}_{sc}$ as defined above, the MCSD problem is to devise an overlay switching scheme which minimizes the average channel switching delay with the receiver successfully switching to the new channel.

Measurement studies have revealed the channel switching delay mainly arises from the bootstrap process [16]. Traditional channel switching protocol arranges the threads of bootstrap and streaming in sequence. We can see those two threads are independent with each other, which leads to the feasibility of parallelizing the threads of boot strap and streaming. In light of that, we propose a more time-efficient protocol to leverage such characteristic. In an intuitive scheme, peers can proactively start bootstrapping in other channels, including getting authorization and retrieving list of peers in the channel while viewing current channel, so that it will be ready to switch channel. As we know, a typical live streaming application, such as PPLive and UUSee, may accommodate hundreds of channels [17]. It is not practical for each peer to personally maintain bootstrapping in all the other channels due to the expensive message overhead.

Towards a feasible solution for this problem, our protocol, NAP, suggests a distributed agent-based bootstrap, where each channel maintains a set of agents responsible for bootstrapping in the channel for external peers. Agents in channel

C_i, denoted as $\mathbf{A}_i$, are selected peers with predicated long lifetime, more storage and communication capabilities in the overlay. They collect the bootstrap information about available peers in the channel with periodical update, and then exchange such information with agents in other channels. It is worth mentioning that when updating, only changed information is collected and distributed to other agents. As shown in Figure 2, on the background of current streaming, peer p, after joining the network, registers at one of the agents $a \in \mathbf{A}_i$ in its current channel C_i and sends a the request for proactive bootstrap to other channels $\bigcup_j C_j$, where $j \neq i$. Agent a buffers and periodically forwards such requests to the agents in the other channels. Those agents in other channels authenticate and authorize p's join request. Once the request is approved, they notify a. Given the bootstrapping information stored at a as $\mathbf{I}_a$, a then sends p a list of peers $\bigcup_j D_p^j$ in the other channels, where $D_p^j \subset \mathbf{I}_a$ and j corresponds to channel C_j. Since then, a updates such bootstrapping information if change happens. Bootstrapping data sent to p contains only a partial set of peers stored in a's list for each channel. A partial set of peers is sufficient as long as the available bandwidth in each channel can fully serve p's streaming. In addition, an agent distills the bootstrapping peers with superior bandwidth and lifetime expectation from ordinary peers in its list to quickly serve the viewer in the initial period of streaming. Moreover, the agents pre-schedule the downloading plan for data chunk partition and streaming for future viewers, and coordinate with each other to build the preventive connections for peers represented by them. We call such distilled peers in the bootstrapping data for peer p as *bootstrapping peers*,denoted as B_p^i for channel C_i. When p switches to a new channel C_j, it requests a service to the agent in C_j and simultaneously request streaming from ordinary peers in D_p^j. The agent forwards the service request to the bootstrapping peers in its management for a quick assistance in the initial streaming period. After bootstrapping peers starts streaming, they keep looking for other ordinary peers to take over their streaming job to p. So the agent can recycle back the bootstrapping peers. By way of this proactive bootstrapping, channel switching delays are reduced significantly.

To reduce the message overhead for updating data on agents, bootstrapping peers in each channel can work only for the initial streaming period T and during that time they handover the streaming to other ordinary peers they found, called *handover peers*. It is reasonable to evaluate T as the expected time that bootstrapping peers need to find handover peers. Because bootstrapping peers can be reused in this way, agents do not need to update the bootstrapping peers as long as they are still in the channel. To reduce the influence from channel switching of bootstrapping peers, strategy of view-upload decoupling (VUD) can be applied with NAP, where what a peer uploads is independent of what it views [3]. Similar to agent selection, bootstrapping peers should have predicted long lifetime and more bandwidth resources. We further discuss the properties of bootstrapping peers and agents in Section 4.

Furthermore, given a peer p that is not new in the network, i.e., p has been viewing some channel in the network, when switching channel, it only registers at the new agent a but does not send bootstrap request since it has done this previously. Bootstrapping data has time stamps. On p's registration, a compare the time stamp t_a on its bootstrapping data with t_p that is stored in p. If $t_a > t_p$, a updates the bootstrapping data to p.

3.2. On Reducing Delay from Peer Churn

In this section, we discuss the protocol to reduce the delay from the other type of churn, i.e., peer churn. Peers can randomly join or leave the overlay of current channel. For most P2P live streaming applications, peer departures have a greater detrimental effect on the delay than new peers arrivals. Peer departures may lead to streaming interruption to their downstream peers. In the paper, we focus on reducing the delay from peer departure.

We call the departure-induced delay as *recovery delay*, which occurs when downstream peers restore their streaming connections. We denote the recovery delay delay as $\overline{d}_{re}$. It can be carried out that

$$\overline{d}_{re} = \overline{d}_p + \overline{d}_{sc}. \tag{2}$$

Comparing with switching delay in Equation (1), notice that recovery delay has no $\overline{d}_A$ and $\overline{d}_{ls}$ components since the peer has already been authorized and owns the peer list in current channel. We now define the problem:

Definition 2: **Minimum Departure-induced Delay Problem (MDD problem):** The MDD problem is to devise an overlay resilience scheme which minimizes the recovery delay during peer departure.

Similar to MSCD problem, we can observe the feasibility of parallelizing the threads of streaming and recovery in this problem. Thus, we can reduce the delay by keeping a proactive bootstrap to the overlay of current channel. This can be accomplished by a simple modification on the protocol described in Section 3.1. Specifically, given an agent a for peer p that is viewing channel C_i, $\mathbf{I}_a$ stored on a is extended to include its bootstrapping data to the current channel C_i. Likewise, D_p^i is stored and updated in peer p as well.

3.3. Management of the Agents

The agents exchange information with each other. A resilient way to realize these exchanges is a gossip-based method. Due to the page limit, we do not focus on how to implement an optimal gossip plan. In addition, an agent may

leave the network, so peers may not reach the agent when they call for a service to the agent. To alleviate this problem, heart-beat signals are exchanged among agents to timely monitor the existence of the agent. In Section 5, we detail the setting of heart-beat signal frequency. Once an agent departure is detected, neighboring agents of the departed agent initiate a process to select new agent. Neighboring agents exchange the copies of bootstrapping information. So when they select new agent, they put back such information that was stored in the departed agent to the new agent.

4. Modeling and Analysis

In this section, we formally model and analyze NAP with queuing theory. In contrast to the traditional Client/Server paradigm and the general P2P file sharing application, the high churn rate and live streaming restriction impose more challenge in modeling a P2P streaming scheme. In the following, we develop a queuing model for NAP considering the failure and restore processes of agents and bootstrapping peers. Although this queueing model is developed for NAP, our study indicates it can also be generally applied to other preemptive schemes after simple modification.

We call the newly arrived peers who did not view any channels *new peers* to the network. For peers that have been viewing channel content, we call them *existing peers*. We model the arrival of new peers to a channel by a Poisson process. In detail, suppose the expected number of new peer arrivals in one time unit is λ_n, then the probability that there are exactly k arrivals of new peers in t time units is equal to

$$f(k;\lambda_n;t) = \frac{(\lambda_n t)^k e^{-\lambda_n t}}{k!}.$$

Likewise, we model arrivals of existing peers from any other channel by Poisson process. Suppose the expected number of existing peers switching from C_i to C_j in one time unit is $\lambda_{i,j}$, the probability that there are exactly $k_{i,j}$ of such arrivals in time units is expressed by

$$f(k_{i,j};\lambda_{i,j};t) = \frac{(\lambda_{i,j} t)^{k_{i,j}} e^{-\lambda_{i,j} t}}{k_{i,j}!}.$$

Theorem 1: Given the expected lifetime of peers on channel C_j as $1/\mu_j$, the expected number of viewers on C_j, denoted as $\overline{N}_j$, is $(\sum_i \lambda_{i,j} + \lambda_n)/\mu_j$ in steady state, where $i \neq j$.

Proof:

The sum of two independent Poisson variables also follows Poisson distribution. More precisely, if $X_1 \sim$ Poisson(λ_1) and $X_2 \sim$ Poisson(λ_2), then $(X_1 + X_2) \sim$ Poisson($\lambda_1+\lambda_2$). Applying this rule iteratively, we have that the arrival rates on Channel C_j follows Poisson($\sum_i \lambda_{i,j} + \lambda_n$).

The P2P streaming system is modeled as an $M/G/\infty$ system here. Thus, according to the queuing theory, the expected number of viewers is given by

$$\overline{N}_j = (\sum_i \lambda_{i,j} + \lambda_n)/\mu_j, \text{ when } i \neq j \tag{3}$$

Thus, the theorem follows. □

Suppose the lifetime of bootstrapping peers, denoted by L_s, follows an exponential distribution, whose probability density function (PDF) can be expressed by

$$f_{L_s}(t;\gamma) = \begin{cases} \gamma e^{-\gamma t} & \text{if } t \geq 0, \\ 0 & \text{if } t < 0, \end{cases}$$

where $1/\gamma$ is the expected lifetime of bootstrapping peers.

As we know, agent will collect the bootstrapping information in its channel and periodically update it. To simplify the problem, we assume the update frequencies on all agents are as same as F. Define the probability that a bootstrapping peer is in the network when it is called to serve other peers as *availability*.

Theorem 2: The expected availability of bootstrapping peer is $(1 - e^{-\gamma/F})F/\gamma$.

Proof: From the lifetime distribution of bootstrapping peers, we can carry out the probability that it will stay for at least time t by

$$\begin{aligned} \Pr\{L_s \geq t\} &= 1 - F_{L_s}(t;\gamma) \\ &= e^{-\gamma t}, \end{aligned} \tag{4}$$

where F_{L_s} is the cumulative distribution function (CDF) of L_s.

As we know, the exponential distribution is memoryless, which means its conditional probability obeys

$$\Pr\{L_s \geq T_R + t | L_s > T_R\} = \Pr\{L_s \geq t\}. \tag{5}$$

Let T_R be update time when an agent is checking if the bootstrapping peer is still in the network. If some bootstrapping peers has left the network, an agent replaces them with new ones. According to Equation (5), once a bootstrapping peer is in the network on the update point, the probability that it stays for at least time t is as same as that of a newly joined bootstrapping peer. Thus, we can apply the same Equation (4) for all bootstrapping peers.

It is obvious that the time of a service call for a bootstrapping peer, i.e., t in Equation (4), is uniformly distributed in the time interval $[T_R, T_R + T_a]$ where $T_a = 1/F$. Accordingly, the expected availability of a bootstrapping peer on a service call, denoted as $\overline{A}_v$, is

$$\overline{A}_v = \frac{1}{T_a} \int_{T_R}^{T_R+T_a} \Pr\{L_s \geq T_R + t | L_s > T_R\} \mathrm{d}t$$

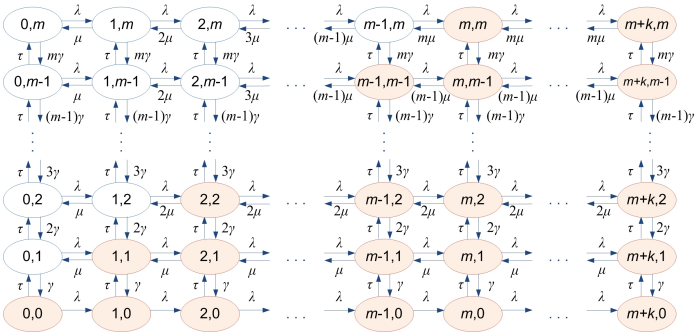


Figure 3: M/M/$m/m+k$ model

$$
\begin{aligned}
&= \frac{1}{T_a}\int_0^{T_a} \Pr\{L_s \geq t\}\mathrm{d}t \\
&= \frac{1}{T_a}\int_0^{T_a} e^{-\gamma t}\mathrm{d}t \\
&= \frac{1-e^{-\gamma T_a}}{\gamma T_a} \\
&= \frac{(1-e^{-\gamma/F})F}{\gamma}.
\end{aligned}
$$

Thus, the theorem follows.

□

Corollary 1: Suppose the $1/\gamma_a$ is the expected lifetime of agents and F_a is the frequency of heart-beat signal to check if the agent is still in the network. The expected availability of an agent $A_a$is $(1-e^{-\gamma_a/F_a})F_a/\gamma_a$.

Suppose that agent can store at most m bootstrapping peers and serve up to $m+k$ new peer requests, where m and k are both non-negative integers. To facilitate the modeling of NAP, we make the following assumptions: 1) the arrivals of new viewers to the agent follow a Poisson process, denoted as Poisson(λ) where λ is the expected arrival rate; 2) the service time of a bootstrapping peer, i.e., the time it takes to hand over the current streaming to other peers, follows an exponential distribution, denoted as Exponential(μ) where $1/\mu$ is the expected service time; 3) the arrivals of new bootstrapping peers to the agent follow Poisson(τ) where τ is the expected arrival rate.

Accordingly, we can model the bootstrapping system on each agent by an M/M/$m/m+k$ queue with bootstrapping peer failure and repair. Figure 3 illustrates our multi-level model, where color-shaded states means available bootstrapping peers are all busy. Given a state (a,b), a represents the number of viewers that are served or waiting to be served by bootstrapping peers, and b describes the number of available bootstrapping peers. To simplify the modeling, we assume a bootstrapping peer can only serve one peer at a time, i.e., $J=1$. For other numbers of J, this model can also work well after simple modifications on b and state transition parameters.

We sequentialize the levels, i.e. rows, in M/M/$m/m+k$ from bottom to top, beginning with 0. Thereby, b^{th} *level* consists of states that have b available bootstrapping peers. Because bootstrapping peers are distilled with superior lifetime expectation from general peers, it is reasonable to assume the arrival rate of general peers is much higher than the departure rate of bootstrapping peers, i.e. $\lambda \gg \gamma$. Thus, we can closely approximate the steady-state probabilities by the following strategy. First, we isolate out each level in state graph as a single 1-level model, analyze it for the steady-state probabilities, and then replace each row by a single state in the original model, analyze this single-column model and carry out the joint probabilities from both models. Figure 4 shows the isolated model on b^{th} level. After replacing each row by a single state in M/M/$m/m+k$ model, we can obtain Figure 5, where each state (b) represents the number of available bootstrapping peers.

Figure 4: Isolated model on b^{th} level

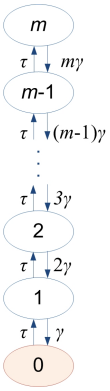


Figure 5: Single-column model

Theorem 3: The probability that a working bootstrapping peer is immediately available for a service call is

$$\sum_{b=1}^{m}\sum_{n=0}^{b-1} \frac{(\lambda/\mu)^n(\tau/\gamma)^b}{n!b![\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i+\sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof: First, we analyze the isolated 1-level model on each row. On the b^{th} level, the probability that a working bootstrapping peer is immediately available once a viewer calls for a service is the sum of probabilities on unshaded states, including states $(0,b),(1,b),$...and(b,b). To solve the steady-state probabilities of the b^{th}-level model, we use the local balance on each state, i.e. the flow of probability into a state equals the flow out of the same state. Let the probability on state (n,b) be $\Pr(n,b)$. Thus, we have the following table of state equilibriums.

State	Equilibrium
$(0,b)$	$\lambda \Pr(0,b) = \mu \Pr(1,b)$
$(1,b)$	$\lambda \Pr(1,b) = 2\mu \Pr(2,b)$
$\vdots$	$\vdots$
(b,b)	$\lambda \Pr(b,b) = b\mu \Pr(b+1,b)$
$(b+1,b)$	$\lambda \Pr(b+1,b) = b\mu \Pr(b+2,b)$
$\vdots$	$\vdots$
$(m+k-1,b)$	$\lambda \Pr(m+k-1,b) = b\mu \Pr(m+k,b)$

By solving the equilibriums in the table, we can carry out the probability on state (n,b) by

$$\Pr(n,b) = \begin{cases} \frac{1}{n!}(\frac{\lambda}{\mu})^n \Pr(0,b) & \text{if } n \le b, \\ \frac{1}{b^{n-b}b!}(\frac{\lambda}{\mu})^n \Pr(0,b) & \text{if } n > b. \end{cases} \quad (6)$$

Second, we replace each row by a single state in the original $\mathrm{M}/\mathrm{M}/m/m+k$ model, and analyze this single-column model. Obviously, the model in Figure 5 is a $\mathrm{M}/\mathrm{M}/m/m$ queue. Applying the same method, we can obtain

$$\Pr(0) = \frac{1}{\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i};$$
$$\Pr(b) = \frac{1}{b!}(\frac{\tau}{\gamma})^b \Pr(0).$$

Next, we apply normalization equation, i.e. $\sum_{i=0}^{m+k} \Pr(i,b) = \Pr(b)$. Therefore, the following can be carried out

$$\begin{aligned} \Pr(0,b) &= \frac{\Pr(b)}{\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i} \\ &= \frac{(\frac{\tau}{\gamma})^b \Pr(0)}{b![\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]} \\ &= \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}. \end{aligned} \quad (7)$$

So in general, the probability that a working bootstrapping peer is immediately available for service, denoted as A_s, can be obtained by

$$\begin{aligned} A_s &= \sum_{b=1}^{m} \sum_{n=0}^{b-1} \Pr(n,b) \\ &= \sum_{b=1}^{m} \sum_{n=0}^{b-1} \frac{(\lambda/\mu)^n (\tau/\gamma)^b}{n!b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}. \end{aligned}$$

Thus, the theorem follows. □

Theorem 4: The probability that a viewer will be rejected when the agent already has $m+k$ service requests is

$$\sum_{b=0}^{m} \frac{(\lambda/\mu)^{m+k}(\tau/\gamma)^b}{b^{m+k-b}(b!)^2[\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof: The probability that a viewer will be rejected, denoted as $\Pr(R)$, is the sum of probabilities on the right-most states, i.e.,

$$\Pr(R) = \sum_{b=0}^{m} \Pr(m+k,b).$$

Applying the results in Equation (6) and (7), we can have

$$\Pr(R) = \sum_{b=0}^{m} \frac{(\lambda/\mu)^{m+k}(\tau/\gamma)^b}{b^{m+k-b}(b!)^2[\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

□

Theorem 5: The probability that all bootstrapping peers in an agent has left the network is

$$\frac{1}{\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i}$$

Proof: The probability that all bootstrapping peers are offline, denoted as $\Pr(L)$, is the sum of probabilities on the bottom states, i.e.,

$$\begin{aligned} \Pr(L) &= \sum_{n=0}^{m+k} \Pr(n,0) \\ &= \Pr(0) \\ &= \frac{1}{\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i}. \end{aligned}$$

The theorem follows. □

Theorem 6: The probability that an agent is idle, i.e. no viewer is now requesting bootstrapping service to it, is

$$\sum_{b=0}^{m} \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof: Similarly, we can deduce the probability that an agent is idle, denoted as $\Pr(I)$, is the sum of probabilities on the left-most states, i.e.,

$$\Pr(I) = \sum_{b=0}^{m} \Pr(0,b)$$

$$= \sum_{b=0}^{m} \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

The theorem follows. □

Lemma 1: Let the expected number of viewers that are served by the agent be $\overline{n}_v$, including those that are being served or waiting to be served. We have

$$\overline{n}_v = \sum_{b=0}^{m} \sum_{a=1}^{m+k} a \Pr(a, b).$$

Lemma 2: The throughput of the peers that is served by the agent is

$$\sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a, b) + \sum_{a=b}^{m+k} b\mu \Pr(a, b)).$$

Proof: Throughput, denoted as x, is the number of viewers that are served in an unit time, i.e. cumulative service rate based on the service rate on each state. To obtain the expected throughput, we should sum up the multiplication of service rate on each state and the probability of the state, i.e.,

$$x = \sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a, b) + \sum_{a=b}^{m+k} b\mu \Pr(a, b)).$$

□

Theorem 7: The expected waiting time of the viewer, denoted as $\overline{d}_w$, i.e. the time a peer waits to be served by a bootstrapping peer, is

$$\frac{\sum_{b=0}^{m} \sum_{a=1}^{m+k} a \Pr(a, b)}{\sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a, b) + \sum_{b}^{m+k} b\mu \Pr(a, b))} - \frac{1}{\mu}$$

Proof: According to Little's law, the total service delay, is $\overline{n}_v/x$, which is the sum of $1/\mu$, i.e. the expected time a peer spends in service, and $\overline{d}_w$. Combining the results from Lemma 1 and 2, we can carry out

$$\overline{d}_w = \frac{\overline{n}_v}{x} - \frac{1}{\mu}$$
$$= \frac{\sum_{b=0}^{m} \sum_{a=1}^{m+k} a \Pr(a, b)}{\sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a, b) + \sum_{b}^{m+k} b\mu \Pr(a, b))} - \frac{1}{\mu}.$$

□

Suppose that a peer will contact j agents for bootstrapping services to avoid agent departure or busy status. To simplify the complexity, we assume there is no bootstrapping agents shared among these agents. The probability that a viewer successfully obtains the service from bootstrapping peers can be expressed by

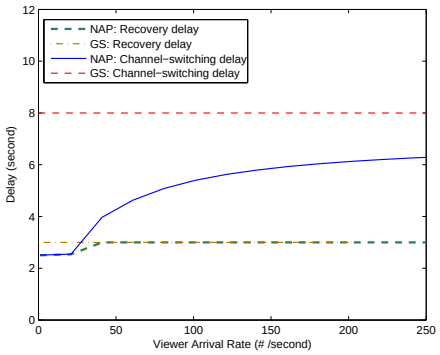


Figure 6: NAP v.s. General Scheme when viewer arrival rate changes.

$$\Pr(S) = (1 - (1 - \overline{A}_a)^j))(1 - \Pr(R)^j).$$

Theorem 8: The expected churn-induced delay based on our scheme, denoted as $\overline{d}$, can be carried out by

$$\overline{d} = \Pr(S)(\overline{d}_w + \overline{d}_f + \overline{d}_{pv}) + (1 - \Pr(S))\overline{d}_b, \qquad (8)$$

where $\overline{d}_b = \overline{d}_p + \overline{d}_{sc}$ is the bootstrapping delay by contacting ordinary peers in the channel for connections, including the communication delay and the delay to get service from ordinary peers, $\overline{d}_f$ is the sum of delays between viewer contacting an agent and agent forwarding the service request to bootstrapping peers, and $\overline{d}_{pv}$ is the expected transmission delay from bootstrapping peer to the viewer.

We do not include the time of retrieving ordinary peer list from agents in the channel because this process runs preventively in the background of viewing current channel and makes no delay when peer switches the channel. Additionally, it is worth mentioning recovery delay has a shorter $\overline{d}_{sc}$ than channel-switching delay because the viewer in recovery has already exchanged the chunk information with some other peers still in the network.

5. Performance Evaluation

In this section, we evaluate NAP against previous general scheme in terms of delay via theoretical experiments and simulation study.

5.1. Theoretical Evaluation

Based on the results from Section 4, we can numerically explore the performance of NAP by tuning some key parameters to the system.

To evaluate the performance of our scheme against the general scheme, we compare the delay occurring in NAP, expressed by Equation (8), with channel switching delay in

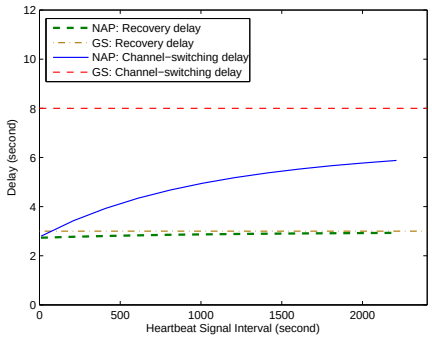


Figure 7: NAP v.s. General Scheme when heartbeat signal interval changes.

the original scheme, i.e., Equation (1) and recovery delay in Equation (2). Besides $\overline{d}_A + \overline{d}_{ls}$, we may find the major difference between $\overline{d}_w + \overline{d}_f + \overline{d}_{pv}$ and $\overline{d}_b$, where NAP saves the time by leveraging agents to pre-schedule the downloading plan and utilize the pre-selected bootstrapping peers to provide a quick and fully streaming service in the initial streaming period.

We abbreviate the general scheme as *GS*. In the first experiment, we compare NAP with GS with varying rate of viewer arrivals to the an agent. According to [16], we set the typical properties of a P2P streaming system as following: 1) the expected time a bootstrapping peer serves a viewer before handing it over to other peers in the network is 6 seconds; 2) the expected lifetime of a bootstrapping peer is 10 minutes; 3) the expected arrival rate of a bootstrapping peer is 1/20; 4) the maximum number of bootstrapping peers that an agent can manage is 100; 5) the number of viewers waiting for the service from bootstrapping peers, if they're not occupied in serving other viewers, is 10; 6) the expected lifetime of an agent is 10 minutes; 7) the frequency of heartbeat signal to check if agent is still in the network is one per 60 seconds; 8) $\overline{d}_A = 0.5$ second; $\overline{d}_{ls} = 0.5$ second; $\overline{d}_p = 1$ second; $\overline{d}_{sc} = 6$ seconds; $\overline{d}_{pv} = 0.5$ second; and $\overline{d}_f = 2$ seconds. For recovery delay, we change $\overline{d}_{sc}$ to 2 seconds since viewers in recovery has already contacted some peers for chunk information before interruption occurs. From Figure 6, we can observe that NAP significantly outperforms GS in terms of channel switching delay. Especially when the arrival rates are lower than 40, almost half of the delay can be avoided. With the increase of viewer arrival rate, the number of idle bootstrapping peers decreases, which leads to the increase in delay. For recovery delay, NAP is better than GS when the arrival rates are lower than 40. When arrival rates goes up, the performance NAP almost converges to that of GS. This is because viewer waiting time is longer when bootstrapping peers are busy. We should notice viewer also contacts ordinary peers simultaneously when they contact the agent for streaming service. This strategy make the worst-case performance of NAP at least equal to GS. To increase the performance, we can add more agents in the channel so as to keep the viewer arrival rate low.

Now, we compare NAP with GS with the change of heartbeat signal frequency. For a better illustration, we use the reciprocal of frequency, i.e. heartbeat signal interval. We keep the setting as the last experiment except that viewer arrival rate is fixed to 25. From Figure 7, we can also see NAP generally outperforms GS in terms of channel switching delay. As of recovery delay, the difference is very close, about 12%. As we know, heartbeat signal is to check if the agent is still in the network, so a larger interval, i.e. less frequency, will increase the probability that an agent has left the network when a viewer contacts it. In that case, agent will use ordinary peers in the startup process, which saves less time than bootstrapping peers. Moreover, it is interesting to see when the heartbeat signal interval is the same as the expected lifetime of an agent, i.e. 10 minutes, NAP can save 43% in channel-switching delay.

From the above theoretical analysis, we observe NAP significantly outperforms GS. To improve the performance of NAP, we can add the number of agents and increasing the heartbeat signal frequency.

5.2. Simulation Experiments

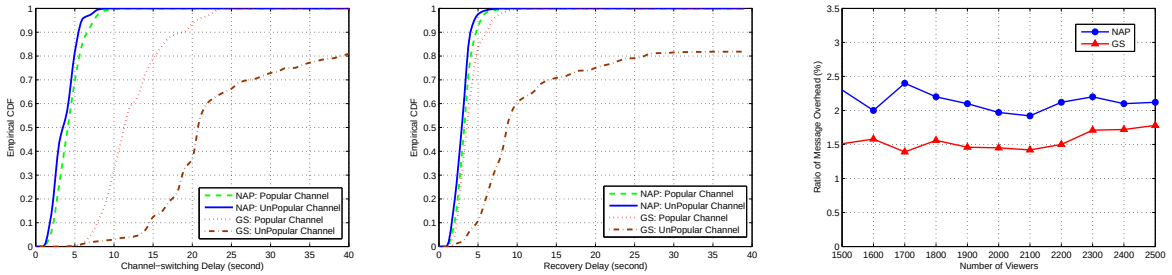
In this section, we compare the performance of NAP and GS via multiple simulations. Our simulation builds on P2P simulator [18], which was enhanced to support switching behaviors among multiple channels.

We simulate a live streaming session of 300 Kbps. From previous studies, we know that network bandwidth exhibits rich diversity [19]. Based on this, we set the upload capacity among peers as shown in Table 1. In our simulation, we set the peak channel popularity follow a Zipf distribution, which is presented in [16]. The streaming system is organized into a mesh structure based on previous work [2]. Peers come to the system in a Poisson process. After viewing a channel for period, peers may depart from the system, or switch to other channel with a probability proportional to the channel popularity. There are initially 50 streaming channels and 2,500 peers in the system.

Upload Capacity	Percentage of Peers
200 Kbps	30%
1.0 Mbps	50%
2.0 Mbps	15%
10.0 Mbps	5%

Table 1: Upload Capacity Distribution

In Figure 8a, we plot the empirical CDF of channel-switching delay under NAP and GS. *Popular channel* means switching to a channel with more than 100 viewers, while



(a) CDF of Channel-switching Delay (b) CDF of Recovery Delay (c) Ratio of Message Overhead

Figure 8: Performance comparison between NAP and General Scheme (GS).

unpopular channel means switching to a channel with less than 10 viewers. From the figure, we observe NAP has much less channel-switching delay than GS in both popular and unpopular channels. Under GS, the delay in popular channel is significantly shorter than in unpopular channel. In contrast, NAP has a slightly better performance in unpopular channel than popular channel. It is because that under GS, bandwidth resource in unpopular channel can easily starve with less peers serving the channel, while under NAP, bootstrapping peers may be allocated to the unpopular channel with more than required bandwidth resources (higher than the actual popularity distribution). In Figure 8b, we illustrate similar experiments for recovery delay. It seems the results show a similar observation with the experiments for channel-switching delay except popular channel under GS has a closer CDF to the CDF curves under NAP.

Figure 8c shows the ratio of message overhead to the video traffic. We can see NAP has a slightly higher message overhead ratio than GS. This result indicates NAP can be widely used in the streaming system without significant message overhead more than GS.

6. Conclusion

In this paper, we develop NAP, an agent-based scheme for reducing the churn-induced delays. Our scheme proposes preventive connections to all channels. Once an actual connection is requested, time is saved in retrieving bootstrapping information and obtaining authorization as well as authentication. Our scheme also suggests an idea of using agents that facilitate the bootstrapping process in channel switching and peer recovery. We analyze the scheme's performance by a queuing model. The experimental results indicate NAP can significantly reduce the churn-induced delays, especially the channel-switching delay.

References

[1] Y. Liu, L. Guo, F. Li, and S. Chen, "A case study of traffic locality in internet p2p live streaming systems," in *ICDCS '09*, 2009.

[2] F. Huang, B. Ravindran, and V. A. Kumar, "An approximation algorithm for minimum-delay peer-to-peer streaming," in *Peer-to-Peer Computing '09*, 2009.

[3] D. Wu, Y. Liu, and K. Ross, "Queuing network models for multi-channel p2p live streaming systems," in *INFOCOM '09*.

[4] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari, "Will IPTV ride the peer-to-peer stream?" *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 86–92, June 2007.

[5] D. Wu, C. Liang, Y. Liu, and K. Ross, "View-upload decoupling: A redesign of multi-channel p2p video systems," in *INFOCOM '09*.

[6] X. Hei, Y. Liu, and K. Ross, "IPTV over P2P streaming networks: the mesh-pull approach," *Communications Magazine, IEEE*, vol. 46, no. 2, pp. 86–92, February 2008.

[7] D. Ren, Y.-T. Li, and S.-H. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *INFOCOM '08*.

[8] C. Wu and B. Li, "rstream: Resilient and optimal peer-to-peer streaming with rateless codes," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, no. 1, pp. 77–92, Jan. 2008.

[9] Q. Wang, L. V. amd Klara Nahrstedt, and H. Khurana, "Identifying malicious nodes in network-coding-based peer-to-peer streaming networks," in *INFOCOM '10*.

[10] K. Xu, M. Zhang, J. Liu, Z. Qin, and M. Ye, "Proxy caching for peer-to-peer live streaming," *Computer Networks*, 2009.

[11] L. Guo, S. Chen, and X. Zhang, "Design and evaluation of a scalable and reliable p2p assisted proxy for on-demand streaming media delivery," *Knowledge and Data Engineering, IEEE Transactions on*, May 2006.

[12] Y. Kim, J. K. Park, H. J. Choi, S. Lee, H. Park, J. Kim, Z. Lee, and K. Ko, "Reducing IPTV channel zapping time based on viewer surfing behavior and preference," in *IEEE Broadband Multimedia Systems and Broadcasting*, 2008.

[13] H. Joo, H. Song, D.-B. Lee, and I. Lee, "An effective iptv channel control algorithm considering channel zapping time and network utilization," *Broadcasting, IEEE Transactions on*, vol. 54, no. 2, pp. 208–216, June 2008.

[14] H. Fuchs and N. Farber, "Optimizing channel change time in iptv applications," in *IEEE Broadband Multimedia Systems and Broadcasting*, 2008.

[15] X. Liu, H. Yin, C. Lin, and C. Du, "Efficient user authentication and key management for peer-to-peer live streaming systems," *Tsinghua Science & Technology*, vol. 14, no. 2, pp. 234 – 241, 2009.

[16] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *Multimedia, IEEE Transactions on*, vol. 9, no. 8, Dec. 2007.

[17] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: Refocusing on servers," in *INFOCOM '08*.

[18] M. Zhang, "Peer-to-peer streaming simulator," http://media.cs.tsinghua.edu.cn/~zhangm/download.

[19] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 6, pp. 1447–1460, Dec. 2008.