An Incentive Protocol for Distributed Dynamic P2P Video-on-Demand Streaming

Wenbin Tang
The University of Hong Kong
Email: wbtang@cs.hku.hk

Xiaowei Wu
The University of Hong Kong
Email: xwwu@cs.hku.hk

T-H. Hubert Chan
The University of Hong Kong
Email: hubert@cs.hku.hk

***Abstract*—P2P file streaming has become very popular in online video sharing. Under the video on demand (VoD) setting, peers in the network may be interested in different portions of the same video. It is a new challenge to distribute the server load to peers under the VoD setting while at the same time maintaining the streaming performance, i.e., low *latency* and good *fluency*. Our approach is to use techniques in social recommendation to spread information on which portions are currently popular. However, peers might not always reveal the truth, because of privacy issues or selfish behavior.**

In this paper, we describe and discuss a synchronized large-scale P2P VoD system in which each peer can only communicate with one other peer in one *round*. We assume the video to be streamed is divided into M consecutive chunks and only one chunk can be transmitted due to network bandwidth every communication. We decentralize the whole system such that each peer has no extra information about the network or how other peers behave, and can only communicate with its own neighbors independently without the help of tracker to obtain robustness. Moreover, the model we consider is fully dynamic: peers leave and join the network frequently.

We show by experiment that even under the bounded connection, bounded transmission and distributed setting, our protocol ensures that almost all peers in the dynamic P2P VoD network can achieve low latency and good fluency in different kinds of network topologies. We also analyze the streaming performance when peers behave differently (selfish vs. unselfish). We show that peers have little incentive to be selfish in our protocol, which means that our protocol is in a sense self-enforcing.

I. Introduction

The peer-to-peer approach has been adopted heavily for content distribution in recent years. Popular P2P file downloading systems like BitTorrent and Emule have proven the efficiency of P2P assisted downloading. The P2P technology is later applied to live streaming systems like PPLive to significantly reduce the server load. Moreover, robustness is achieved in those systems since the peers interact with each other independently under decentralized controls. Recent attention is drawn to a new kind of service named P2P video-on-demand (VoD) in which peers may be interested in different portions of the same video. P2P VoD streaming systems (PPLive, PPStream and UUSee) were developed to alleviate the server load. Peers in those systems contribute their own upload bandwidths. Hence a large population of peers can be served at the same time [5], [6].

The research was partially supported by the Hong Kong RGC grant HKU719312E and the PROCORE grant F-HK31/11T.

It was posted by Cisco in 2012 that by 2017, $80 \sim 90\%$ of the global consumer Internet traffic will be caused by P2P file sharing, live streaming and VoD [3]. Among these applications, VoD traffic has been growing rapidly for the past few years. It is challenging to design mechanisms to evenly allocate the server load to all peers since different peers may have different viewing positions [6], [8].

Usually a *tracker* is placed in the system to maintain connections and also help distributing server load evenly to all peers. However, the role of tracker actually weakens the robustness of the system by damaging the decentralization. In a distributed P2P VoD system, peers only make limited number of connections [12], [13] to existing peers and only communicate with connected peers (neighbors) without any global vision of the whole system [4]. Peers leave and join the system frequently and hence the system is highly dynamic. To support P2P VoD, it is commonly assumed that peers in those systems contribute their local storages (usually 1GB) to help other peers [6]. Moreover, peers in those P2P VoD systems will unselfishly help their neighbors by not only contributing their upload bandwidths, but also by disseminating their neighbors' requests to possible source providers.

Former research on BitTorrent [7] and also on P2P VoD systems [10] were performed to analyze how incentive protocols encourage unselfish behaviors. However, all those results suggest that it is difficult (or sometimes impossible) to design incentive protocols in which peers are motivated to dedicate their bandwidths to help each other. Most of those protocols [7], [10], [9] are equipped with rewarding mechanisms to reward unselfish peers. It is hence an interesting question whether an incentive protocol can be effectively designed so that without any rewarding mechanism, all peers will behave unselfishly to maximize their own (expected) profits. Those protocols are also called *self-enforcing* since peers have little incentive to be selfish.

We analyze in this paper a fully distributed and dynamic P2P VoD system in a round-by-round manner. Assuming that all peers in the system have bounded bandwidths, each peer (including the server) is only allow to make constant number of connections and only initiate one communication with its neighbor in each round. Based on statistics from [6], we assume the starting position of each peer is independently identically distributed following some distribution on the whole video. We design a distributed protocol in which the bandwidths of all peers are efficiently used such that all peers' requests are likely to be disseminated and responded. We designed a scheme called *Collaborative Request Selection*

(CoRS) to help peers disseminating urgent requests and hence can use the whole network bandwidth to help peers under bad situation, i.e., waiting for a long time. Our protocol (with CoRS adopted) is applied to several different network models to show the performance, which is measured by the *latency* and the *fluency* of each peer. We show by experiments that our protocol guarantees that almost all peers achieve low latency (short waiting time) and good fluency (smooth viewing) under all tested network topologies.

To analyze how unselfish behavior (disseminating requests from their neighbors) is encouraged in our protocol, we compare the network and peer performances when some peers behave selfishly. Our experiments show that (1) if only one peer behaves selfishly, then its latency and fluency are not much better than before; (2) the network performance is still good if only a small fraction of peers behave selfishly. Given above, it is easy to see that peers have little incentive to be selfish because being selfish is not guaranteed to be profitable. Hence even without any rewarding mechanism equipped, our protocol is still incentive by implicitly encouraging unselfish behaviors.

II. Model and Protocol Description

We analyze the distributed video streaming problem in a network consist of N peers in a synchronized setting. At the beginning, the owner of the video (which we called the *server*) cuts the video into M *chunks* such that each chunk can be transmitted in a short period of time, which is called one *round*. The video streaming procedure is proceed in a round-by-round manner. With bounded download and upload bandwidth, the server would like to distribute the load caused when all N peers are trying to watch the video at the same time.

A. Requirements

Streaming Requirement. Each peer joins the network for the purpose of watching the video. Hence, they all require the chunks of the video to be received in the correct order, which means that there will be lots of communication collisions when the bandwidth is bounded.

Bounded Connections. Each peer only has a constant number C of neighbors. During the streaming process, unless some connections fail, no more connections are allowed to be made.

Distributed Communication. We assume that each peer (including the server) is only allowed to communicate with its own neighbors in the network, except the first time it joins the network or any of the current connections is no more available. Under the distributed setting, each peer has no global vision of the network and can not directly communicate with the server unless it is a neighbor of the server.

Bounded Bandwidth. Since each peer has a bounded download and upload bandwidth, it is only allowed to make one communication with one of its own neighbors every round and respond to a limited number of communication requests. For each communication established, only one chunk can be transmitted in each direction, which means that at most 2 chunks will be transmitted in each communication. If the number of communication requests some peer u receives is beyond the limit, only some of them (according to the limitation) will be made successfully. Note that since at most N communications are established every round, the bandwidth of most of the peers are not fully used, which is a common observation in the real network.

Video-on-Demand. To support VoD, we assume that a constant fraction of peers will start watching the video at some random positions. In that case, each peer u (with starting position $start_u$) can start watching the video only after chunk $start_u$ is received.

B. The Model and the Protocol

Let N be the number of peers and M be the number of chunks. At round $t = 0$, only the server has M chunks indexed by $[M] = \{1, 2, \ldots, M\}$, all other peers have no chunk and are waiting to receive the chunks in the correct order. Assume that each peer makes C connections with existing peers (which means a maximum number of C out-neighbors). Recall that at most 2 chunks are transmitted in each communication.

The set of chunks each peer u has can be indicated by a bit set of length M (*chunk bit-set*) $\{\beta_1, \beta_2, \ldots, \beta_M\}$ such that $\beta_i = 1$ if u has chunk i and 0 otherwise. Let $start_u \in [M]$ be the starting position of peer u. Let p_u be the current downloading position of peer u, which is the smallest index of the chunks that is after the current viewing position of u and not received. Let s be the number of rounds to consume (watch) one chunk. Note that s indicates the ratio between the downloading speed and the consuming speed. We call a peer u *started* if it has received chunk $start_u$ and *finished* if it has obtained all desired chunks $\{start_u, start_u + 1, \ldots, M\}$.

How Peers are Connected: Network Topology

There are at most N peers in the network at the same time. Each peer will perform a peer-searching procedure by the following rules when it first joins the network.

1) BTC-Network: each peer will choose C neighbors uniformly at random from existing peers and make connections when the first time it joins the network. During the streaming, if some peer leaves the network, then each of its in-neighbors will choose a backup neighbor uniformly at random independently from all existing peers.
2) PA-Network [2]: each peer will choose C neighbors from existing peers with probabilities proportion to their degrees and make connections when the first time it joins the network. During the streaming, if some peer leaves the network, then each of its neighbors will choose a backup neighbor from all existing peers with probabilities proportion to their degrees independently.
3) Twitter-Network [11]: the network is extracted by removing peers with degree 1 and taking the largest connected component (nearly $50,000$ peers and $260,000$ edges) from the graph of Twitter [11]. For streaming, if any peer leaves, the new peer joining the network will take its place and preserve the same set of links. To reduce the maximum degree of the network to a constant, we assume that each peer only select a constant number C of neighbors uniformly at random and only communicate with those neighbors during the streaming.

We consider a fully dynamic network.

1) At the beginning there is only one peer (server) and every round λ peers join the network and make connections until there are N peers.
2) Each peer can quit the downloading any time during the procedure independently with probability q every round. To ensure that a constant fraction of peers watch the video until it is over, we set $q = \Theta(\frac{1}{M})$.
3) Each peer that has finished the watching will leave the network immediately.
4) Assuming a fixed size network, when a peer leaves the network, a new peer will join the network and make connections as described above.

Where Peers Start: Video on Demand

Since not all peers will watch the video from the very beginning, we assume that a constant fraction ϵ of peers will perform random seek when it joins the network. (According to [6], we set $\epsilon \approx 0.5$) We use the same distribution to model the starting positions of random seekers as in [13]. The starting position of each random seeker (peer that starts watching from a random position instead of the beginning) is identically independently distributed by the following exponential distribution: $Pr[start_u \geq i] = e^{-\frac{3i}{M}}, \forall i \in [M]$, which is equivalent to $Pr[start_u = i] = \frac{3e^{-\frac{3i}{M}}}{M(1-\frac{1}{e^3})}, \forall i \in [M]$.

How Peers Communicate: 4 Phases

Each peer will choose one of its neighbors uniformly at random every round and make communication. The communication is divided into the following 4 phases (some details will be discussed later).

1) **Establishment.** At the beginning of each communication, the initiating peer u will send out the communication request to the chosen neighbor v. However, since the bandwidth of each peer is bounded, each peer can only accept a bounded number of communication requests (chosen uniformly at random). Hence with some probability, the communication request might be ignored by v and u can not make any extra communication this round.
2) **Exchanging Chunk Requests.** Once the communication is established, the two peers u and v will exchange the information about what chunks they want. Each peer preserves a list of chunks it desires to obtain (*requests list*). We assume that only a bounded number k of requests from the requests list can be chosen and sent during each communication.
3) **Exchanging Chunks.** Based on the chunks each of them has and their neighbour's requests, each peer chooses a chunk it has that is requested by its neighbor for transmission (if there is any).
4) **Updating Local Information.** At the end of each communication, based on the chunks and the requests each peer receives, the chunk bit-set and the requests list will be updated accordingly.

Note that for each instance of communication, only one communication request (confirmation) and k chunk requests (each of size $O(\log_2 M)$) will be sent in each direction, which compared to the size of chunk (usually 500 Kb) are negligible.

How Peers Behave: Selfish vs Unselfish

Note that p_u will be contained in the requests list of u as long as u is not finished. Based on whether a peer disseminates its neighbors' requests, we consider 2 types of peers.

Unselfish Peers. Peer that tries to help its neighbors by storing and passing its neighbors' requests if they are more urgent. Mechanism is designed and adopted to distinguish urgent requests from non-urgent requests. Hence, if an unselfish peer u has enough buffer, it will not necessarily choose p_u as a request but probably urgent requests from its neighbors. The requests from the neighbors of an unselfish peer u will be included in u's own requests list;

Selfish Peers. Peer that only tries to get chunks after its downloading position, which means that it will never ask for or download any chunks that are not chunks after its downloading position. The requests list of those peers only contains chunks after their downloading position, i.e., $\{p_u, p_u + 1, \ldots, M\}$.

C. Measurements

The performance of the above protocol can be measured by the *latency* and the *fluency* of each peer.

Latency. The latency l_u for peer u is defined to be the number of rounds peer u spends on waiting for chunk $start_u$. In application, short latency implies that peers can start watching the video soon after they join the network.

Fluency. For each peer u that is started, we call a round *bad* for u if u has consumed all chunks it has and is waiting for the next chunk to come; *good* if u has some unwatched (fraction of) chunk. Let b_u be the number of bad rounds and g_u be the number of good rounds, then the fluency of peer u is defined to be $f_u = \frac{g_u}{g_u+b_u}$, which is the ratio between the number of good rounds and the total number of rounds after u receives chunk $start_u$. Note that $f_u \in (0, 1]$ for each peer u and larger f_u indicates less time is spent on buffering. Also note that $g_u = (M - start_u + 1) \times s$ if u leaves the network after it finishes watching the video.

III. Collaborative Request Selection

In this section, we introduce the *Collaborative Request Selection* (CoRS) scheme that is adopted to disseminate requests. We define a data structure called REQUEST to represent a request. Each peer will have a prioritized list of requests. Each request is generated and stored with the following 4 fields:

1) **ID.** The requested chunk's ID.
2) **Urgent.** A boolean flag indicating whether this request should be put into high priority. When a peer has no chunks to watch, an urgent request is generated for the requested chunk. Otherwise, a normal request with $urgent = false$ is generated.
3) **Hop Limit.** The limit on the number of hops a request is allowed to travel through before it expires. The hop limit is set to h when the request is generated and decreased by 1 every round. Hop limit is used to control the number of copies of one request and reduce the number of requests in the whole network.
4) **Time Stamp.** The time stamp of a request is set to be the round by which the requested chunk should

be received so that the streaming can be carried out fluently. For example, if peer u is starting to watch chunk w_u, then the time stamp for the request for chunk p_u should be set to *current round*$+(p_u - w_u)s$.

Note that for each round, if peer u is not finished, then a new request for p_u will be generated (with hop limit equals to h) and merged into u's requests list. The requests list of each peer is sorted according to the following rules:

1) urgent request has higher priority then non-urgent request;
2) if two requests have the same urgent field, then earlier time stamp implies higher priority;
3) if two requests have the same urgent and time stamp fields, then lower hop limit implies higher priority.
4) if two requests have the same urgent, time stamp and hop limit fields, then smaller ID implies higher priority.

Consider an established communication between u and v.

Selection of Requests. Since each peer can only choose at most k requests for transmission, if the requests list contains more than k requests, the top-k requests with highest priority will be chosen and sent out.

Selection of Chunks. Suppose k prioritised requests are sent from u to v, then v will check the requests one by one following the priority to find the first chunk that v has and is requested by u for transmission.

Updating Local Information. At the end of each communication, since each peer may receive new chunks and new requests, the local information (chunk bit-set and requests list) should be updated accordingly. It is obvious that the chunk bit-set should be updated to indicate the acquisition of the new chunks.

The requests list of each peer u is updated as follows.

1) The hop limits of all newly arrived requests and requests in the requests list are decreased by 1.
2) If u is not finished, a new request for chunk p_u will be generated.
3) All existing requests will be merged to form a new requests list: if several requests contain the same ID, then merge these requests into one, set its hop limit to the highest one, the time stamp to the lowest one and urgent if one of them is urgent.
4) After that, delete from the requests list all requests with hop limit 0 and requests for already received chunks.

IV. Performance Evaluation

Based on the real world statistics, we fix our experiment parameters as follows in this section.

A. Basic Settings

According to a recent report published by *Akamai* [1] at July 2013, the global average connection speed had climbed to 3.3 Mbps, and the percentage of networks with an average speed > 4 Mbps is more than 50%. Considering that Hong Kong has the world's highest average peak connection speed (65.1 Mbps), we assume in the P2P VoD streaming network, the download bandwidth (and hence the upload bandwidth) is bounded by 6.6 Mbps.

We assume that the video to be streamed is encoded in flv format with 360×640 resolution (360P), 25 frames per second and 320 Kilobit per second (Kbps). Let the size of one chunk be fixed to 512 Kb and assume that the video is cut into $M = 1000$ chunks. Then the whole video will last for $\frac{512*1000}{320} = 1600$ seconds, which is approximately 26.7 minutes. Note that it takes $\frac{512}{320} = 1.6$ seconds to consume one chunk. Assume that each peer can only initiate one communication and respond to a maximum number of 4 communication requests every round. Since only one chunk can be downloaded in each communication, at most 5 chunks can be downloaded in each round. Since it takes $\frac{5*512}{6.6*1024} \approx 0.379$ second to finish the transmission of 5 chunks, we fix the length of one round to be 0.4 second, which means that each chunk will be consumed in $s = \frac{1.6}{0.4} = 4$ rounds.

We set $N = 2^{14} = 16384$ to indicate the maximum number of peers watching the same video at the same time. Since it is easy to achieve good streaming performance in a small size network, we set N large enough to show how a large population of peers can be served at the same time by our protocol. Considering the balance between robustness and efficiency of the streaming protocol, we set $C = 4$ for the BTC-network, PA-network and Twitter-network by which peers are connected.

As defined before, we set the fraction of random seekers to be $\epsilon = 0.5$ and probability of leaving during the watching to be $q = \frac{1}{s \times M} = 1/4000$. We set $k = 20$ be the maximum number of requests each peer can send in each communication; $h = 10$ be the hop limit of each request. By Section III, each request is of size roughly $\lceil \log_2 M \rceil + 1 + \lceil \log_2 h \rceil + 36 = 51$ b, which implies that an extra $k \times 51 = 20 \times 51 = 1020$ b (≤ 1 Kb) package will be transmitted in each communication. Since each chunk is of size 500 Kb, the extra data is negligible.

B. Viewing Position Distribution

With parameters fixed as above, the viewing position of each peer when it first joins the network is chosen under the following distribution (Fig. 1).

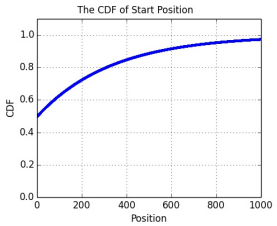

Fig. 1. The CDF of the starting position

As we can see, about $\epsilon = 0.5$ fraction of peers start from the beginning and other peers start from some random positions with distribution defined in [13]. Note that the distribution matches Figure 6 (the CDF of the viewing position of movies) of [6] quite well. The assumption [13] that an $e^{-3} \approx 0.05$ fraction of peers' downloading position is $M = 1000$ is also used by our experiments.

C. Protocol Performance: Latency and Fluency

We show our protocol performances under three different network topologies in this section. For space reason, we will only discuss issues in the BTC network model in detail. However, our experiments show that similar results can be obtained in the PA-network model and Twitter-network model. From now on, we will be focusing on the BTC-network.

Since the network is fully dynamic, if a peer u joins the network at round t_1, starts watching at round t_2 and eventually leaves the network at round t_3, then the latency of u can be calculated by $l_u = t_2 - t_1$ and the fluency of u is the fraction of good rounds between t_2 and t_3: $f_u = \frac{g_u}{t_3 - t_2}$. We run the P2P VoD system under three different network topologies for $30,000$ rounds and collect all l_u's and f_u's of each peer u that leaves the network before round $30,000$. By our experiment, more than $400,000$ peers leave before we stop collection and hence we have more than $400,000$ data points (l_u, f_u).

The latencies of peers in the BTC network that we collected are distributed as follows (Fig. 2). As we can see from Fig. 2, more than 99.6 percent of peers have latency ≤ 10 rounds and 99.8 percent of peers have latency ≤ 50 rounds. Recall that each round lasts for 0.4 second, hence $l_u \leq 10$ implies that peer u starts watching the video 4 seconds after it joins the network and $l_u \leq 50$ implies that peer u starts watching 20 seconds after it joins the network. Moreover, the fraction of peers that are suffering from large latency ($l_u \geq 100$) is less than 0.1%.

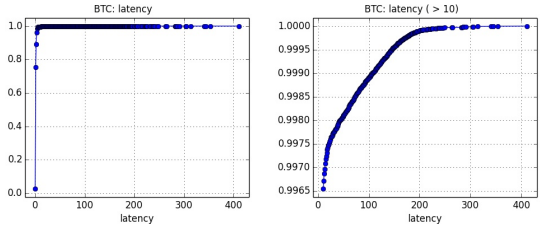

Fig. 2. The CDF of the latency in BTC

The fluencies of peers are distributed as follows (Fig. 3). If peer u has fluency f_u and leaves the network after watching chunk end_u, then u spends $\frac{1-f_u}{f_u} \times (end_u - start_u) \times s \times 0.4$ seconds on waiting during the streaming process (excluding the latency). Note that $f_u = 1$ means that there is no interrupts during the streaming until peer u leaves the network. Recall that the video lasts for about 1600 seconds, which can be taken as an upper bound for $(end_u - start_u) \times s \times 0.4$. Hence if $f_u \geq 0.99$, then totally less than 16 seconds are spend on waiting; if $f_u \geq 0.97$, then totally less than 50 seconds are spend on waiting.

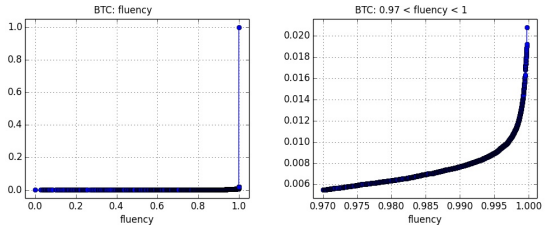

Fig. 3. The CDF of the fluency in BTC

By the above figures, we can see that about 98 percent of peers obtain perfect fluency ($f_u = 1$), which means that almost all peers in the network do not even need to wait for a single second after the first few seconds (latency), and more than 99.2 percent of peers have good fluency ($f_u \geq 0.99$). The fraction of peers that are under bad situation ($f_u \leq 0.97$) is only 0.6%.

It is shown by the above two figures that our protocol makes sure that with probability at least 0.996, a peer in the network only need to wait for less than 4 seconds before watching; with probability at least 0.992, a peer in the network only need to wait for less than 16 seconds during the streaming process. Note that if peer u is not satisfied by the network situation (i.e., $l_u \geq 100$ or $f_u \leq 0.97$), it can simply leave the network and then join again. Our experiment suggests that with high probability, the situation will be good enough if u rejoins network. Since we assume that each peer (including the server) only has constant number of neighbors and (in expectation) can only download less than 2 chunks every round, the experiment result suggests that our protocol actually achieves a great succeed in distributing chunk requests and helps the P2P VoD system to achieve a good efficiency.

D. Selfish vs. Unselfish

We analyze how the protocol performance will be affected when some peers are selfish. Remind that a selfish peer u will always sent $\{p_u, p_u + 1, \ldots, p_u + k - 1\}$ as requests to its communication partners and will never include any requests from its neighbors into its own requests list.

Is Being Selfish Profitable?

We analyze the difference between being selfish and being unselfish by comparing a peer's latency and fluency under those two behaviors, given that all other peers behave unselfishly. To make sure that the only differences between the peer's latency and fluency are caused by the peer's own behavior (instead of other peers' behaviors or random choices), we fix the randomness used by all peers in those two cases to be the same. After fixing the randomness, we can make sure that the network topologies are the same in those two cases and moreover, all random choices made by each peer on communications are the same.

Our experiment is carried out by choosing a random peer u and then record u's latency l_u and fluency f_u if u behaves unselfishly; record u's latency l'_u and fluency f'_u if u behaves selfishly. We see the difference of latency by points (l_u, l'_u) and difference of fluency by points (f_u, f'_u) after repeating the random selection of peers independently for enough times. We put all collected (l_u, l'_u) points into one figure to indicate the how the latency is affected by being selfish. Note that if all data points are close to the base line $l_u = l'_u$, then it means that being selfish makes little difference; if most of the data points are above the base line, then it means that being selfish actually hurts the latency. We also put all collected (f_u, f'_u) points into one figure to indicate the how the fluency is affected by being selfish. Similarly, if all data points are close to the base line $f_u = f'_u$, then being selfish makes little difference; if most of the data points are below the base line, then it means that being selfish actually hurts the fluency.

Since we already know that all l_u's are close to 0 and all f_u's are close to 1, to make the figure less biased, we collect data points (l_u, l'_u) and (f_u, f'_u) for different ranges of l_u and f_u independently and then put all data points together into

one figure. The following two figures show the differences of latency and fluency after being selfish.

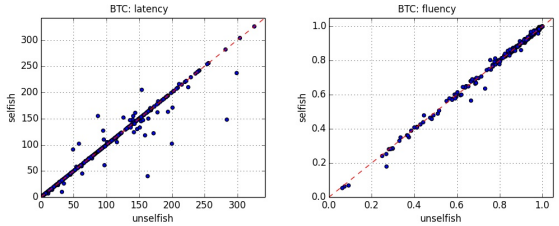

Fig. 4. Differences of latency and fluency in BTC

As shown in Fig. 4, for latency, being selfish is slightly profitable if originally (unselfish case) the peer has small latency (≤ 50). However, since most of the data points with $l_u \geq 50$ are at or above the base line, the conclusion is that if originally peer u has latency $l_u \geq 50$, then (with high probability) being selfish is not beneficial. More importantly, data points above the base line indicates that in some cases being selfish will actually be worse since the latency is increased. If originally peer u has large latency, then a possible implication is that u is not connected to peers under good situation. In that case, lots of urgent requests will be generated by u's neighbors. Since u does not help its neighbors to disseminate the urgent requests, those requests (compared to the case when u is unselfish) are less likely to be responded. Hence every round, the request from u is less likely to be disseminated and responded since it will be competing with many other urgent requests that are not yet responded. Hence, if u is selfish, then it is only guaranteed be profitable if originally u has small latency. However, if that is the case, u does not even need to be selfish. Since being selfish may actually be harmful, to maximize u's own benefit, it is wise to be unselfish.

For the case of fluency, since all most all data points are really close to the base line, we can see from Fig. 4 that actually it makes very little difference to be selfish. Hence by giving consideration to both fluency and latency, it is not profitable to be selfish and each peer has little incentive to be selfish, which means that our protocol, especially with the CoRS mechanism, is self-enforcing.

How Will Selfish Peers Affect the Network?

Instead of analyzing the protocol performance on one selfish peer, we analyze how the network performance is affected if many peers are selfish at the same time. Note that if only a constant number of peers are selfish, the network performance will not be affected. Here we assume that each peer becomes selfish after it joins the network with probability p and unselfish with probability $1 - p$, independently for all peers. By varying the value of $p \in [0, 1]$, we can see how will selfish peers affect the network. Note that if we collect data from enough number of peers, then by measure concentration, the fraction of selfish peers will be close to p.

Our measurement in this case is the whole network performance. We measure the average latency and average fluency for each value of p by collecting data points while running the P2P VoD system for $30,000$ rounds. For each p, before the collection is stopped, roughly $400,000$ peers leave the network and the data points are calculated by taking average over all latencies and over all fluencies. The following figure shows how the average latency and average fluency are changed when we vary p from 0 (all unselfish) to 1 (all selfish).

It can be seen from Fig. 5 that as the fraction of selfish peers increases, the network performance becomes worse. Compared to the case when all peers are unselfish, in which all peers only need to wait less than 1 second in average for the first chunk, when the fraction of selfish peers becomes 0.6, in average every peer has to wait twice the length of time (3.5 seconds) before start watching. Similarly, while all peers (in expectation) only stop for less than 1.6 seconds when $p = 0$, if $p = 0.6$, then in average each peer will be stopped for buffering for 4.6 seconds. In extreme case when all peers are selfish, in average every peer need to wait more than 6 seconds for the first chunk and (in total) has 6 seconds stopped during watching the video. However, when the fraction of selfish peers is less than 30 percent, our protocol still guarantees that the average latency is upper bounded by 2 (which means waiting 0.8 seconds for the first chunk) and the average fluency is lower bounded by roughly 0.999 (which means being stopped for 1.6 seconds during watching the video).

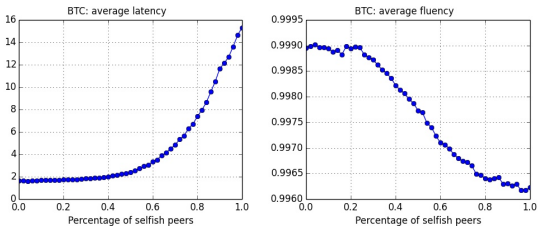

Fig. 5. Average latency and fluency in BTC

ACKNOWLEDGMENT

We sincerely appreciate the discussions with Chenzi Zhang, Weiqing Liu and Jian Zhao on the model and protocols.

REFERENCES

[1] Akamai. The state of the internet. 2ND QUARTER, 2013 REPORT.

[2] Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, 2004.

[3] Cisco. Cisco visual networking index: Forecast and methodology, 2012-2017. white paper published on Cisco website, 2012.

[4] Bram Cohen. Incentives build robustness in bittorrent, 2003.

[5] Cheng Huang, Jin Li, and Keith W. Ross. Can internet video-on-demand be profitable? In *SIGCOMM*, pages 133–144, 2007.

[6] 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 *SIGCOMM*, pages 375–388, 2008.

[7] Michael Piatek, Tomas Isdal, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? (awarded best student paper). In *NSDI*, 2007.

[8] Kai Wang and Chuang Lin. Insight into the p2p-vod system: Performance modeling and analysis. In *ICCCN*, pages 1–6, 2009.

[9] Tin-Yu Wu, Wei-Tsong Lee, Nadra Guizani, and Tzu-Ming Wang. Incentive mechanism for p2p file sharing based on social network and game theory. *Journal of Network and Computer Applications*, 2013.

[10] Weijie Wu, John C. S. Lui, and Richard T. B. Ma. Incentivizing upload capacity in p2p-vod systems: A game theoretic analysis. In *GAMENETS*, pages 337–352, 2011.

[11] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *WSDM*, pages 177–186, 2011.

[12] Can Zhao, Xiaojun Lin, and Chuan Wu. The streaming capacity of sparsely-connected p2p systems with distributed control. In *INFOCOM*, pages 1449–1457, 2011.

[13] Can Zhao, Jian Zhao, Xiaojun Lin, and Chuan Wu. Capacity of p2p on-demand streaming with simple, robust and decentralized control. In *INFOCOM*, pages 2697–2705, 2013.