We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Reverse Engineering the Peer to Peer Streaming Media System

Chunxi Li[1] and Changjia Chen[2]
[1]*Beijing Jiaotong University*
[2]*Lanzhou Jiaotong University*
China

1. Introduction

Peer to peer (P2P) content distribution network like BitTorrent (BT) is one of most popular Internet applications today. Its success heavily lies on the ability to share the capacity of all the individuals as a whole. As the first deployed prototype on the real Internet, CoolStreaming (Zhang et al., 2005) for the first time manifests what a great application potential and huge business opportunity it can reach if the content is delivered not only in large scale but also on real-time. Along the way to the large-scale Along the way to such as system, people (Vlavianos et al., 2006) find there is no natural connection between the abilities of mass data delivering and real-time distributing in any protocol. This discovery stimulates people to study how to modify protocols like BT to meet the real-time demand. Since 2004, a series of large-scale systems like PPLive and PPStream have been deployed in China and all over the world, and become the world-class popular platforms. Many research reports on them also mark their success.

However, most existing works are descriptive. They tell about how such a system works and how to measure it, but do not pay much effort to explain why. In this chapter, we take a different route. We seek to better understand the operation and dynamics of P2P systems at a deeper level of detail. We split our understanding objective into the following sub-objectives 1) understand the working principle through the communication protocol crack, 2) comprehend the streaming content-delivery principle, 3) locate the measurable parameters which can be used to evaluate the system performance; 4) understand the P2P network through the models of startup process and user behavior, and analyze the engineering design objectives. The requirements for reaching those goals are as follows. 1) the research must be driven by mass measured data of real network. 2) for us, the measuring platform must be suitable to the normal access situation like the home line. 3) datasets must be available in terms of scalability, quality and correctness of information. 4) the process of reversing engineering should be well designed with ease to set up the analysis, ease to interpret the results and ease to draw conclusions from the presented results.

However, the road towards reaching our research goals is full of challenges. On this road, many new findings are reported, many original problems are presented, and many design philosophies are discussed for the first time. Because all P2P streaming systems so far are proprietary without any public technical documentation available, the fundamental "entry

point" of the analysis is to crack the system protocol, and then develop measurement platform to access to the system legally; next, based on the mass raw data, we investigate and study the user/peer behaviors, especially the startup behaviors which are believed to involve much more systematic problems rather than stable stage; at last, the system's performance, scalability and stability are discussed and the design models and philosophy are revealed based on the peer behavior models. The research steps outlined previously in this paragraph are detailed in Sections 3 to 5. In addition, Section 2 presents related work.

2. Related works

Since the deployment of CoolStreaming, many measurement based studies are published. Some useful measurable parameters such as *buffer width*, *playable video* and *peer offset* are defined in (Ali et al., 2006; Hei et al., 2007a, 2007b; Vu et al., 2006), and startup performance is addressed in user perceptive (Zhou et al., 2007). In fact, nearly all the reports assume a small buffer system, which is far from the real one like PPLive that adopts much large buffer to resist network fluctuant. For a system like PPLive, one can no longer simply assume the same situation for both stable and startup peers. Studies on a mixed system of CDN server and peers can help our study. It is shown in (Lou at el., 2007; Small at el.,2006; Tu at el., 2005; Xu at el., 2003) that, there is a *phase-transition point* $C(t)$ at time t in the mixed network, any chunks below $C(t)$ is easy to fetch. The issue like $C(t)$ in P2P steaming media system has never been studied. Besides, data fetching strategies are theoretically discussed in many reports. The algorithms of rarest first and greedy (Zhou at el., 2007) are two extreme strategies arise from BT and a mixed strategy of them is proposed in (Vlavianos at el., 2006; Zhou at el., 2007), while what is the suitable fetch strategy in P2P streaming media system needs to be answered. On VoD system aspect, very few studies (Cheng, 2007; Huang, 2007) based on so-called P2P VoD system were ever seen in 2008, however the target network is far from we discussed at all. The server-based VoD users' behavior is studied in (Yu et al., 2006; Zheng et al., 2005) based on core server's log file, but it is questionable whether P2P user has the same feature. Besides, intuitionally, data-sharing environment and user behavior will influence each other in P2P VoD system unlike in server based VoD system, however no relative research reports that.

3. Signalling crack and network measurement

Reverse-engineering-based protocol crack is the first step. It helps understand the working mechanism in depth, but also makes our large-scale measuring possible by developing network crawler. To the best of our knowledge, the work presented here and in related papers by the same authors and colleagues is the first in the world who succeeded in cracking and measuring all the top popular P2P streaming media systems in large scale.

3.1 Brief description of P2P VoD system

Referring to Fig.1, a typical P2P media streaming system uses few servers to serve large number of audiences (named as *peer*) with both live and VoD programs (Ali et al., 2006; Hei, et al., 2007a; Zhang, et al., 2005). There are significant different design concerns about P2P VoD system and live system: *i).* VoD peer uses much more storage space to cache nearly the whole video in long term than live peer to cache very few latest contents temporarily. Besides, VoD peer may share all the cached contents even if he is in a different channel. (b) P2P live system is of source-driven such that seeder controls the content feeding rate, while

P2P VoD system is of receiver-driven and each peer controls playback rate by himself. Unlike live peer, VoD user has more flexibility to choose different playback patterns, such as skipping, fast forwards and fast backwards.

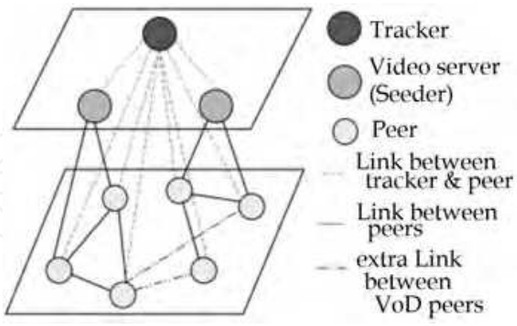

Fig. 1. The system structure

3.2 The communication protocol cracking

In general, the protocol crack is a cycling procedure including following steps:

Network sniffer/measurement: In the first step, performed using a client sniffer, we capture the interactive packets between the local peer and others. We get to know the important protocol messages must be there such as *shake hand* message, buffer map message (*BM*), and peer list message (*peerlist*), based on existing research reports and our experience. By connecting those types of message to the sniffer trace, it is not difficult to distinguish all kinds of message, even though some messages' functions are unknown.

Protocol message guess: Next, we observe each message in different dimensions, including the dimensions of time, channel and peer. For facilitating observation, we use a small software (developed by us) to extract the wanted messages with some query conditions, such as source IP/port, destination IP/port and message type, from the traces. From the extracted records, we can see many regular patterns which help parse the detailed format of each message. Of course, this way doesn't always work well, for the minority of messages can't be explained. So, we don't neglect any available reference information, e.g., we have ever found the fields of total upload/download count and upload/download speed per peer contained in BM based on the information displayed in PPStream client window. In general, we crack more than 80% messages for PPLive, PPStream and UUSee.

Test and Confirmation: In this stage, we analyze and validate the interactive sequences of messages. We guess and try different interactive sequences until the normal peer or tracker gives the right response. At last, nearly all the guesses are confirmed by our successfully and legally access to the real network.

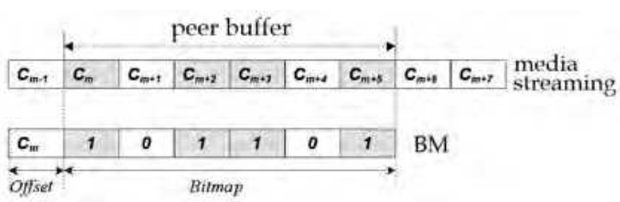

Fig. 2. Buffer and buffer map

Though the crack process, we get to know some principle of the system. In the P2P system, the video streaming is split into many blocks called *chunks*, each of which has a unique ID. In general, *chunk ID* is sequentially assigned in ascending order, i.e. the earlier played chunk has smaller ID as Fig.2 shown. A seeder injects chunks one by one into the network and each peer caches the received chunks in a buffer. Live peer only caches a small fraction of the whole video, while VoD peer caches almost the whole video. A peer buffer is usually partially filled. The downloaded chunks -- the shadow square in Fig.2 are shared, while the empty areas need to be filled by downloading from others. For enabling the key sharing principle between P2P users, a message *BM* is introduced to exchange the buffer information between peers. Referring to Fig.2, for a peer p, its BM contains two parts, an offset f_p and a bitmap. The offset f_p is the oldest chunk ID, i.e., the smallest chunk ID in a buffer. The bitmap is a $\{0,1\}$ sequence, which length indicates the buffer width W_p. In the bitmap, a value of 1, respectively 0 at the i^{th} position start from left to right means that the peer has, respectively has not the chunk with $ID_{offset+i-1}$. Since a peer constantly fetches new chunk to fill its buffer and shifts the expired chunk out of the buffer, the chunk IDs at both ends of the buffer will go forward with time, we name the BM *offset* time sequences of as *offset curve* $f_p(t)$ and the largest chunk ID time sequence in the peer's BM as *scope curve* $\xi_p(t)$. Obviously, the difference between them is the peer's *buffer width* $W_p(t)=\xi_p(t)-f_p(t)$. Usually, $W_p(t)$ fluctuates with time. In addition, we get a very useful finding in tracker *peerlist* message: Different from the *peerlist* of a peer, the tracker *peerlist* has two important extra fields, Tk_{OffMin} and Tk_{OffMax}, corresponding to the buffer head (called *seeder offset* f_{tk}) and buffer tail (called *seeder scope* ξ_{tk}) of the seeder, respectively. Obviously, the seeder's buffer width is $W_{tk}=\xi_{tk}-f_{tk}$. The finding can be proved in next section.

3.3 Network measurement and dataset

Using the cracked protocols, we succeeded for the first time to develop crawlers that measure different P2P streaming media systems in a large scale. The crawler first reads a channel's index file. Then it starts to collect BMs and peerlist messages returned by tracker or peers into a log file as the raw trace for our offline studies, meanwhile we insert a local timestamp into each message. The crawler runs on a PC server (512 kbps ADSL home line, window XP, 2.4 GHz CPU, and 1 GB memory). The VoD crawler trace used in this chart is captured from PPLive on October 26, 2007, and lasts for about 90 min. The live crawler trace is also captured from PPLive during the time period from Apr. 2 to Jul. 15, 2007. With the crawlers, nearly all peers in any given channel can be detected, so that much more properties can be found. However, crawler is incapable of detecting a peer within its very beginning stage because the startup peer doesn't emit any signaling messages to a normal peer/crawler. Thus, a live sniffer trace, which is captured on July 3, 11, 12 and 15, 2007 by using a sniffer tool, is used to analyze the startup progress. We call it an experiment for each client sniffing and the trace totally contains about 2500 experiments.

4. Reverse engineering analysis from a peer's viewpoint

Like the BT system, live peer may play roles of *leecher(watcher)* or *seeder*. A seeder has the complete video, while a leecher hasn't. In a P2P live streaming media system, all peers are

watchers and a few content servers are seeders. On the other hand, a P2P VoD system also contains two roles. However, they are not classified based on whether a peer has a complete file or not. Although most VoD peers do not own a complete video, he can share it once he is online regardless of the viewing channel. In a channel, we name a peer never downloading from others as a *contributor*, and a peer downloading from others as a *watcher*. VoD *watcher* is just like live watcher in many aspects, while VoD *contributor* may not necessarily have a complete file. As a *contributor*, the VoD peer may upload one movie while watching another. A significant difference of a VoD system from a live system is that *contributors* largely outnumber *watchers*. Our measurement shows that about two-thirds peers are *contributors*.

4.1 Live peer behavior in P2P streaming media system

Nearly all existing studies simply assume a stable playback rate. Thus we start with the problem of video playback rate measurement to launch our analysis. Then, we raised the questions of how a peer reaches its stable playback state, and whether and how a peer can keep in good shape.

4.1.1 Playback rate and service curve

Intuitively, the forward BM offset with time t in peer p, noted as $f_p(t)$, is connected to its playback rate. According to our experience, small rate changes are hidden if we were to visualize $f_p(t)$ directly as a time sequence. Instead, a curve of $rt–f_p(t)$ with proper value of playback rate r can make the changes obviously. However, to check every peer's playback rate is a hard job. In practice, each peer has its own playback rate which roughly equals to the system playback rate, otherwise video continuity cannot be ensured. Thus, a system playback rate should be found as a common reference for observing peer offset progress.

We describe the system playback process by a *service curve* $s(t)$. It is reasonable to use the system maximal chunk ID at any time t as $s(t)$, and then playback rate is $r(t) = ds(t)/dt$. For a channel with playback rate variations, the playback rate vs. time should be a piecewise linear function.

The procedure of finding the rate change is similar to the method in estimating the clock skew in network delay measurements. In (Zhang, 2002), people presented "Convex_Hull_L" algorithm and a segmented algorithm, which are denoted as CHU and SCHU respectively in our research, to calculate the network delay. However, referring to Fig.3, the convex envelope (dash line) calculated by CHU fails to reflect the rate changes in medium time scale in our trace 070502. Through slightly modifying SCHU algorithm, we get a new method called Piecewise Line Envelop Approximation (PLEA) (Li & Chen, 2009). The *rate reset time* $\{t_k\}$ and *reset rate* $\{r_k\}$ is simply the turn point and slope of each segment in the piecewise line calculated by PLEA respectively. The key of PLEA is to take convex hull only in small time scale and follow the rate variation in medium time scale. Thus, a parameter named as *follow-up time* Δ is introduced. An observed point will be kept if the time difference between this point and previously saved point is larger than Δ. Unlike SCHU, our segmentation is automatically adjusted during the calculation procedure without pre-assigned or fixed. The square marked line in Fig.3 shows the result of PLEA with Δ=1500s. It

fits the envelope of trace quite well. Comparing PLEA to SCHU in Fig.3, the result of PLEA is much smoother.

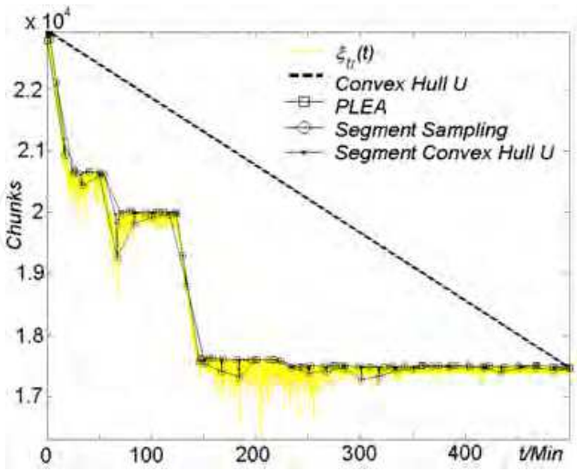

Fig. 3. PLEA v.s. others algorithms

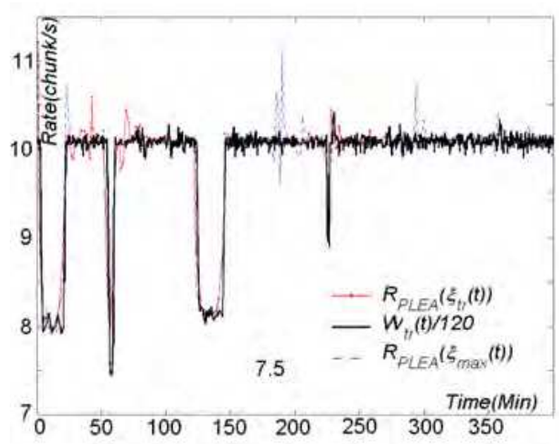

Fig. 4. Comparison of our algorithms

Besides PLEA, we have an occasional but very useful finding during reverse engineering. In PPLive, the seeder's buffer width $W_{tr}(t)$ reported by tracker, which is the difference of seeder's scope minus its offset, is always equals to the product of 120 and current playback rate r, i.e., $W_{tr}(t)=120r(t)$. For validation, we draw all the rate curves calculated from PLEA of tracker scope $\xi_{tr}(t)$ and peers max scope $\xi_{max}(t)$, *i.e.*, $R_{PLEA}(\xi_{tr}(t))$ and $R_{PLEA}(\xi_{max}(t))$, as well as $W_{tr}(t)/120$ in the same trace in Fig.4. All rate curves match well except some individual points. Thus we have following observations: For any PPLive channel, the instantaneous rates deduced from both tracker scope and peer maximal scope equal each other, and they are about 1/120 of the seeder's *buffer width*, i.e., $R_{PLEA}(\xi_{tr}(t))=R_{PLEA}(\xi_{max}(t))=W_{tr}(t)/120$.

Then new questions are naturally raised. Whether has the system took the rate variations into account in design? When rate change occurs, can that lead a peer to restart? All such questions involve a primary problem, what is operating mechanism of a peer, especially in its early stage.

4.1.2 The observation of a peer based on key events

By sniffing many single clients, the typical events in peer startup progress are revealed in Fig.5. For simplicity, we call a startup peer as *host*. The first event is the *registration* message a host sends to the tracker after selecting a channel. We take the registration time as the *reference time* 0. After certain *tracker response time* T_{tk} the host gets a peerlist response from the tracker, which contains a list of online peer addresses and the seeder buffer information (Tk_{OffMin} and Tk_{OffMax}). Next, the host connects to the peers known from the peerlist. Shortly after, the host receives its *first BM* at *peer response time* T_p, and the sender of the first BM is correspondingly called the *first neighbor p*. After that, the host chooses an initial chunk as its start point and begins to fetch chunks after that chunk. We denote the time when a host sends its first chunk request as the *chunk request time* T_{chk}. After a while, the host starts periodically advertising its BM to the neighbors. The time when a host sends its first BM is named as the *host advertising time* T_{ad}. This time breaks the whole start process into two phases: the *silent phase* and the *advertising phase*. Only in the advertising phase, a host can be sensed by an outside crawler. We find that, in a short time period after T_{ad}, host reports an invariant BMs' offsets, which indicates a host is busy in building his buffer so that it's not the time to start to play video. At the time called *offset initial time* T_{off} when the host begins to move the BM offset forward, we think the host begins to drain data out from its buffer for playback. By the way, an oftenly-used *offset setup time* τ_s is defined as the time duration between T_p and T_{off}, i.e. $\tau_s = T_{off} - T_p$. Time points of T_{tk}, T_p, and T_{chk} are all in the silent phase and can only be detected by client sniffer. While after T_{ad}, time points of T_{ad} and T_{off} can be collected by either our crawler or a sniffer. We use both two platforms to measure peer's startup process and use T_{ad} as the common reference to connect both platforms.

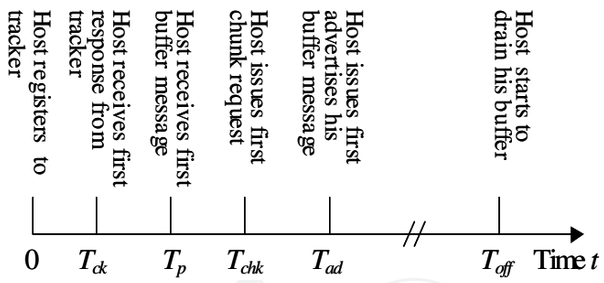

Fig. 5. Events and their occurring time in PPLive

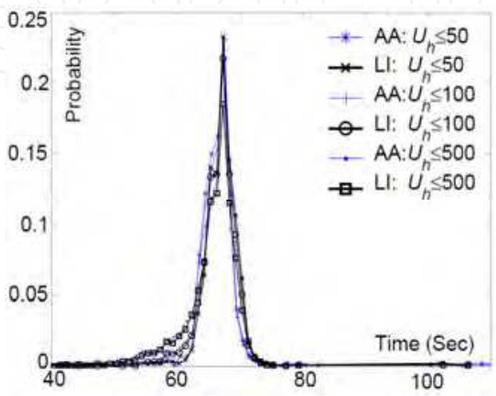

Fig. 6. PDF of τ_s'

Firstly, we measure the events in the silent phase. Statistic based on our 2502 client sniffing experiments shows that the T_{ad} is a multiple of 5s and in most of cases $T_{ad} \approx$5s. Most trackers return their responses within 0.02 seconds. T_p has an evenly distribution in the time interval [0.9s 2s]. Among T_{tk}, T_p, T_{chk} and T_{ad}, no time points are tightly dependent. The average values of T_{tk}, T_p and T_{chk} are 0.058s, 1.419s and 2.566s respectively.

Secondly, we measure the events in advertising phase. The sniffer method is not used because it can only detect limited hosts which are under our control. With our crawler, nearly all the peers in any given channel can be detected, so that much more properties can be found. For each peer p, we get the offset $f_p(t)$ at each discrete time of $\{t_{p,i}\}$. Not every peer caught by our crawler is a host since many peers have already been there before the crawler inquiries them. A principle is used to extract the hosts from our trace, i.e., a host should have an invariable offset in its early BMs and then increase the offsets later. Two problems are involved in inferring the exact value of T_{off} from BM records. First, T_{ad} is often missed out in BM records. In the most cases, a host has been in progress for uncertain time before our crawler queries him. Second, we can't get the exact time when a host starts to drain data from his buffer because the time span between T_{ad} and T_{off} may last for several tens of seconds. We take following measures. *Buffer fill* $U_h(t)$, which is the number of all downloaded chunks in the buffer, i.e., the number of 1s in a BM at time t for a given host h, is used to solve the first problem. We only choose the peer meeting the condition $U_h(t) \leq$ certain threshold as a host, and take the timestamp of its first BM as its *advertising time* T_{ad}. For checking if the threshold introduces biases, we try different thresholds. For the second problem, we take two different methods to estimate that time when a host changes its offset. Let t_1 and t_2 be the timestamps of the earliest two BMs with different offsets $f(t_1) \neq f(t_2)$. One is the simple arithmetic average (AA) $T_{off} = (t_1+t_2)/2$, and the other is the linear interpolation (LI) $T_{off} = t_2 - (f(t_2) - f(t_1))/r$. The relative offset set time τ_s' for a host is calculated as $\tau_s' = T_{off} - T_{ad}$. The probability distribution functions (PDFs) of τ_s' estimated by AA and LI with different thresholds are plotted in Fig.6. The similarity of the results can validate above methods. Therefore, we get peer's *offset setup time* $\tau_s = \tau_s' + (T_{ad} - T_p) \approx$70s where the mean value of τ_s' is about 66s and $T_{ad} - T_p$ is about 5–1.419≈3.6s measured in silent phase. Then, what is the root reason for that constant, why not other values? Let's dig it more deeply.

4.1.3 Model-based observation of peer initial offset selection

We name the peer's first wanted chunk as the *initial offset* θ. We reconstruct a peer startup model in Fig.7 to explain the importance of initial offset. Assuming a constant playback rate r, *service curve* $s(t)$ is a global reference. Assuming a constant seeder buffer width W_{tk}, we have the seeder's offset curve $f_{tk}(t) = s(t) - W_{tk}$ below $s(t)$. The host's first neighbor p's *offset curve* and *scope curve* (of its largest chunk ID) are $f_p(t)$ and $\xi_p(t)$ respectively. Since the number of successive chunks in a buffer indicates how long the video can be played continually, we follow (Hei et al., 2007b) to name that as the *buffer's playable video* $V_p(t)$, correspondingly the *peer's playable video* $v_p(t) = f_p(t) + V_p(t)$, which is also drawn in Fig.7. The initial offset is very important for that, once it, saying θ_h, is chosen at certain time t_h, the host's offset lag $L_h = s(t) - f_h(t)$ is totally determined. As shown in Fig.7, $f_h(t)$ begins to increase after the τ_s, meanwhile $s(t)$ has increased $r\tau_s$. Since the host initial offset lag is $L_\theta = s(t_h) - \theta_h$, its *offset lag* at last is $L_h = L_\theta + r\tau_s$. L_h is the playback lag, but also the possible maximum buffer width. It means θ_h can affect the system sharing environment.

For simplicity, we assume the *initial offset* decision is based on the host's *first neighbor p*. Then, host *h* faces two alternatives -- based on either the tracker or its first neighbor. Seeing Fig.7, at time t_h, host *h* gets values of $s(t_h)=Tk_{OffMax}$ and $f_{tk}(t_h)= Tk_{OffMin}$ from tracker, and values of $\xi_p(t_h)$, $v_p(t_h)$ and $f_p(t_h)$ from its first neighbor *p*. Then the host should choice its θ_h between $f_p(t_h)$ and $\xi_p(t_h)$, beyond which scope no chunk is available. For further explanation, the chunk θ_h will shift out of the neighbor *p*'s buffer at time $t_h+(\theta_h-f_p(t_h))/r$. Large θ_h lets host *h* have more time to fetch this chunk. However, as too large θ_h will lead to a very small *offset lag*, host's *buffer width* maybe not large enough for a good playback performance. So what are the design principles behind the initial offset selection?

We extract the marked points shown in Fig.7 at time t_h from our 2502 experiments, and draw them as a function of sorted experiment sequence in ascending order of W_{tk} and $W_p(t)$ in Fig.8 where we take $f_{tk}(t_h)$ as the horizontal zero reference. The red lines are the seeder's buffer width $\pm W_{tk}=\pm(s(t_h)-f_{tk}(t_h))$. The top one is W_{tk} and the bottom one is $-W_{tk}$. Clearly, PPLive mainly serves two playback rates: 10 chunks/s on the right area and 6 chunks/s on the left area. The *black* '.' and *green* '*x*' stand for ξ_p-f_{tk} and f_p-f_{tk} respectively, the distance between which marks in each experiment is peer p's buffer width $W_p=\xi_p-f_p$. Similarly, the vertical distance between top red '-' and *green* '*x*' is peer p's offset lag $L_p=s-f_p$. Thus, Fig.8 confirms that PPLive takes certain variable buffer width scheme. Furthermore, seeder has a larger buffer than normal peer. The *blue* '*' is hosts relative initial offset lag θ_h-f_{tk}. Obviously, PPLive doesn't adopt a fixed initial offset lag scheme, or else all *blue* '*' would keep flat. Actually the *blue* '*' and *green* '*x*' have a similar shape, which means that *initial offset* may adapt to *first neighbor p's* buffer condition.

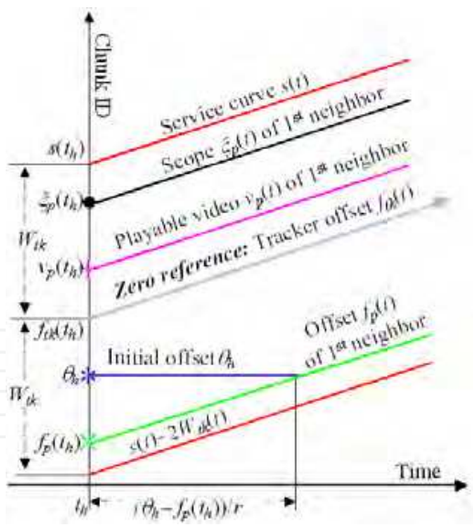

Fig. 7. The startup model

We think certain kind of *Proportional Placement* (PP) strategy (Li & Chen, 2008a) can be introduced to make the decision of *initial offset*. Referring to Fig.8, the distance of the initial offset to its first received BM's offset is somehow proportional to the first neighbor's buffer width $W_p=\xi_p-f_p$ or the first neighbor's offset lag $L_p=s-f_p$. Thus, we guess PPLive chooses the *initial offset* either by $\theta_h=f_p+\alpha_W W_p$ or $\theta_h=f_p+\alpha_L L_p$, where the α_W and α_L are the scale coefficients. Based our measurement, both PDFs of $\alpha_W=(\theta_h-f_p)/W_p$ and

$\alpha_L=(\theta_h-f_p)/L_p$ have the very high peaks at the same coefficient 0.34. The scaled errors of $100(\alpha_W-0.34)$ is shown with the cyan color in Fig.8. It seems that PPLive more likely uses a scheme based on the first neighbor's buffer width since α_W has a more sharp distribution. To check whether the selected *initial offset* θ is easy to download, as well as to evaluate whether PPLive has been designed to make host set its *initial offset* at the most suitable point locally or globally, we have studied the chunk availability. As a result, a host usually receives BMs from 4.69 peers before fetching any chunks. In more than 70% experiments, host can fetch chunks around θ from at least 3 neighbors. It indicates a good initial downloading performance.

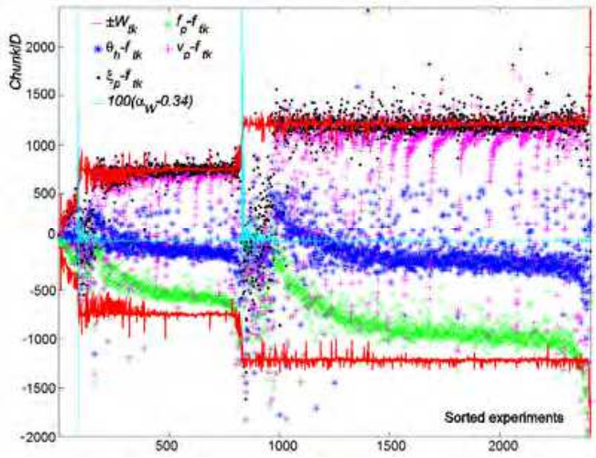

Fig. 8. The measured initial parameters

4.1.4 Model-based peer observation in the startup stage

Once the initial offset is chosen, the host begins to download chunks. We use a simple model to help understand the data fetching process. For any given peer p, the model contains two parts. One is *buffer filling process,* expressed by curves of *buffer width* $W_p(t)$, *playable video in buffer* $V_p(t)$, *and buffer fill* $U_p(t)$ which is the number of all downloaded chunks in the buffer at time t. They reflect a buffer's local conditions, but can't tell the status of peer process in a global sense. The other is *peer evolutionary process* depicted by curves of *offset* $f_p(t)$, scope $\xi_p(t)$, peer *playable video* $v_p(t)$, *download* $u_p(t)=f_p(t)+U_p(t)$ and the reference $s(t)$. Ideally, for a CBR video, all evolutionary process curves should have the same slope equals to the playback rate r. One real progresses of the model can refer to Fig.9. The top line is $s(t)$ as the reference line, the black line at the bottom shows the offset curve $f_p(t)$, and the cyan curve close to $s(t)$ is $u_p(t)$; the solid red curve with mark 'x' is $W_p(t)$, the green curve with mark '*' is $U_p(t)$, and the blue curve with mark '+' is the $V_p(t)$.

Obviously, the downloading procedure contains two kinds of strategies. In Fig.9, both $W_p(t)$ and $V_p(t)$ have a same switch point at ($\tau_{sch}\approx$40s, $C_{sch}\approx$900). We guess, before time τ_{sch}, a peer sequentially fetches chunks from small to large ID, which can be confirmed by the fact of the closeness of $W_p(t)$, $V_p(t)$ and $U_p(t)$ before τ_{sch}. Ideally, the three curves should be the same. However, in real networks, some wanted chunks may not exist in its neighbors or a chunk request maybe rejected by its neighbor. At the switch time point τ_{sch}, the big jump of $W_p(t)$ indicates a fetch strategy change. Therefore, we name τ_{sch} as the *scheduling switch time*. Before

and after τ_{sch}, we call the downloading strategies used by a peer as strategy I and strategy II respectively. A peer fetches chunks sequentially in strategy I, while in strategy II it may always fetch the latest chunk first. At the switch point to strategy II, the chunk ID's sudden increase leads to an enlarged buffer width.

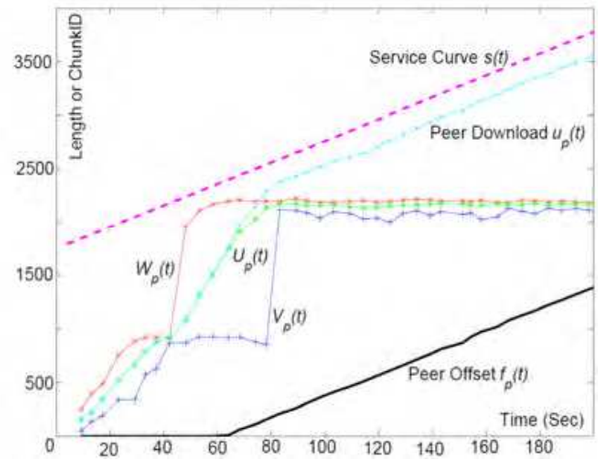

Fig. 9. A PPLive peer's evolution

We believe the downloading strategies switch may be base on certain ratio threshold of buffer filling, and a closer observation can support this guess. As shown Fig.9, BM offset $f_p(t)$ keeps flat in the early 65s. Then, the peer starts to shift the offset forward. Let's see the flat playable video $V_p(t)$ curve duration time [40s, 80s]. We can infer the first flat part in period of [40s, 65s] is for that the peer is downloading the latest chunks according to strategy II. If with the same strategy, the curve of the rest part in period of [65s,80s] should have had sloped downwards. Thus it must have changed the fetch strategy again from strategy II to I, which let peer fetches the most urgent chunks first so as to keep the $V_p(t)$ at certain threshold.

At last, all curves of $W_p(t)$, $V_p(t)$ and $U_p(t)$ converge after a time around 80s, which is named as *convergence time* τ_{cvg}. A sudden big jump in $V_p(t)$ at this time indicates that the last wanted chunk within the buffer are fetched. It proves that the latest chunk is fetched firstly by strategy II in most of the period of $[\tau_{sch}, \tau_{cvg}]$.

The whole progress can be approximated by a set of piecewise linear functions by a threshold bipolar (TB) protocol, which is very simple in its implementation and design philosophy. For a host, when the current $V_p \leq$ a threshold, the urgent task is to download the most wanted chunks, while if $V_p >$ the threshold, the job is switched to help spread the latest or rarest chunks over the network. We have ever observed some other peers' startup procedures in our trace, and all of them can be interpreted easily by the TB protocol.

By further observation, the piecewise line model involves six structure parameters including video *playback rate* r, peer *initial download rate* r_p, fetching strategy *switch threshold* C_{sch}, *offset setup time* τ_s, the *initial offset* θ_p relative to the first neighbor's offset and the *offset lag* W^*. Among them, r and r_p cannot be designed and the rest four can. Assuming a constant r and a constant r_p, based on the superposition principle at the key

points among the piecewise lines, it is not difficult to calculate other key time points, including *scheduling turnover time* τ_{sch} and *convergence time* τ_{cvg}, and then we can draw the piecewise lines. (Li & Chen, 2008b).

We are next interested to better understand the parameters design in PPLive. In order to generalize our discussion we consider all the relative parameters, including C_{sch}, θ_p, W^*, and nearly all buffer progress parameters, to be normalized by playback rate r, and for simplicity, we use the same names for most of the normalized parameters as their originals.

We have known the *offset setup time* $\tau_s \approx 70$s in subsection 4.1.2. For C_{sch}, we use *switch threshold factor* $\beta = C_{sch}/r$ instead of C_{sch}. The calculation of C_{sch} is a litter fussy, referring to Fig.9: *i).* let $C_{sch}=W_p(t)$ just before the first jump of $W_p(t)$; *ii).* Let $C_{sch}=V_p(t)$ just after the first big change in $dV_p(t)/dt$; *iii).* let C_{sch} =mean of $V_p(t)$ on its flat part; *iv).* let $C_{sch}=V_p(t)$ just before the jump of $V_p(t)$. The results of all above methods are plotted in Fig.10, and we have $\beta=90s$. Next, W^* is deduced from our crawler trace. Based on the statistics over total 15,831 peers lasting for at least 5 minutes since they entered stable state, we get a similar result for both normalized buffer width and offset lag relative to $s(t)$. At last, the *relative initial offset* is figured out from sniffer trace. The distribution of W^* and θ_p are shown in Fig.11. Based on our measurement, we have W^*=210s and θ_p = 70s.

Corresponding to the different sort orders of τ_s, τ_{sch} and τ_{cvg}, i.e. $\tau_s<\tau_{sch}<\tau_{cvg}$, $\tau_{sch}<\tau_s<\tau_{cvg}$ and $\tau_{sch}<\tau_{cvg}<\tau_s$, after computation with these design parameters of PPLive, we get three groups of the buffer process, $\Gamma_0=\{\gamma_p: <\gamma_p\leq 1.286\}$, $\Gamma_1=\{\gamma_p: 1.286<\gamma_p\leq 3\}$ and $\Gamma_2=\{\gamma_p: \gamma_p>3\}$, where γ_p is the *normalized download rate* $\gamma_p=r_p/r$. Peers in group Γ_0 face a very poor startup condition. They take very long time to converge and the convergence time spans from 490s (about 8min) to infinite (γ_p=1, never converge). According to our measured γ_p, less than 10%peers belong to this group, while more than 70% peers belong to group Γ_1. Hence Γ_1 is the normal startup situation, and the convergence time is between 490s and 70s. Peers (more than 20%) in Γ_2 are so fast that they have converged before playing the video.

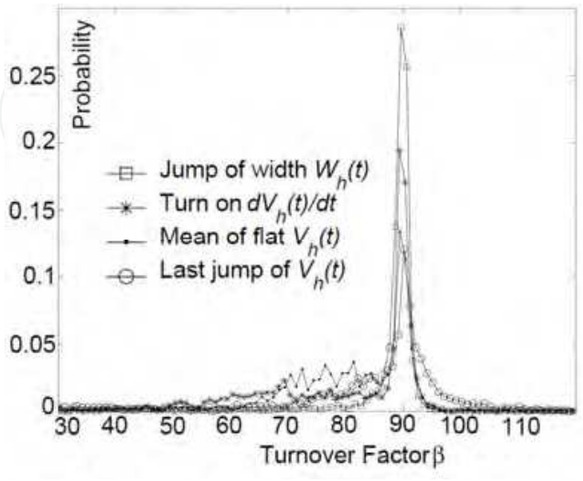

Fig. 10. Probability distribution function of β

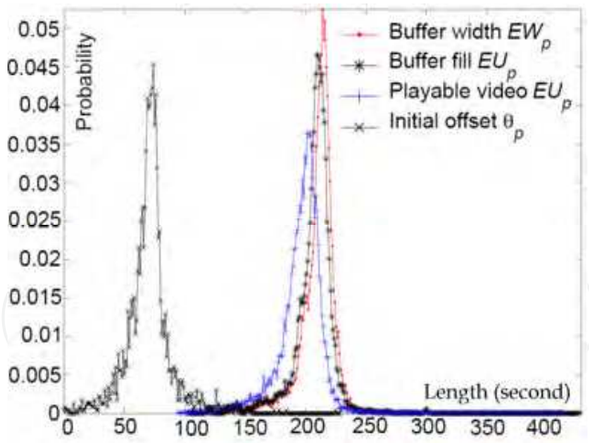

Fig. 11. Probability distribution function of buffer progress (normalized)

In summary, we have described how to approximate peer evolutionary progress based on the six parameters and the so-called design parameters in PPLive. In general, PPLive client has a good startup performance. In next section, we will reveal the systematic concerns behind the parameters design in PPLive.

4.2 VoD user behavior in P2P streaming media systems

In general, A VoD peer can be classified as *contributor* or *watcher* based on whether the number of ones never increases in bitmap of the peer's BM or not during our observation. In our trace, most peers belong to either contributor or watcher. Less than 6% peers even advertised the abnormal all-zero BMs, the bitmap contained nothing. We guess such disordered behavior ascribed to software bugs, e.g. a user deletes his cache file suddenly. We name such those peers as Zpeer. Fig.12 draws the fractions of different peer groups in our measured channel 1. In fact, the rest two measured channel have the similar results. Those curves confirm that contributors always significantly outnumber watchers, and a stationary process can approximate the fractions.

Further, two types of watching modes have been identified. People either watch a movie smoothly until his exit, or see a movie by jumping from one scene to another. We named the former as smooth watching mode and such viewer as smoother, and the latter as the jumping watching mode and that viewer as jumper. Obviously, smoother has continuous 1s in its BM, while jumper has discrete 1s. Table 1 lists the statistics on our trace. We find the majority are smoothers, while the jumpers cannot be ignored. It is different from that "most users always perform some random seeking" (Zheng et al., 2005).

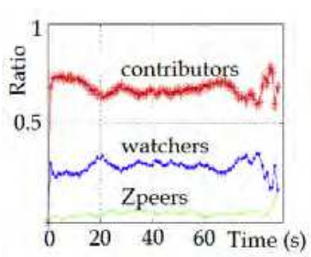

Fig. 12. role ratios in channel 1.

	Contributors				Watchers			
	Smoothers		Jumpers		Smoothers		Jumpers	
	Peers	%	Peers	%	Peers	%	Peers	%
Channel 1	300	70.9	123	29.1	138	87.3	20	12.7
Channel 2	264	86.8	40	13.1	90	90.1	9	9.1
Channel 3	156	73.2	57	26.8	82	78.8	22	21.2

Table 1. Number of smoothers and jumpers

4.2.1 Measureable parameter watching index in user behavior

For quantitative analysis, we introduce *watching index (WI)* to name the position of the last "1" in a BM, which explains how many chunks a smoother has ever watched. Different from definition in (Yu et al., 2006), we use WI to emphasize the aspects of both time and space. As most peers are smoothers, a movie with a larger WI or longer tail in WI distribution in smoothers is usually considered to be more attractive. It means that people watched this movie longer or more people watch the movie. We use probability $p_{WI}(\theta)$ to represent the PDF of WI, which is the fraction of peers whose last "1" in their BMs are at the position θ. Fig.13(a) shows Cumulative Distribution Function (CDF) $F_{WI}(\theta)=\sum_{k\le\theta} p_{WI}(k)$. Obviously, channel 3 and channel 2 were the most and the least attractive respectively. Besides, online time is defined as how long a peer stays in a channel, and Fig.13(b) shows its CDF. Obviously, distributions of WI over all channels are significantly different but their online times are very similar. It indicates that WI is strongly related to the video content, while the contributor's online time is nearly independent of what he is currently contributing.

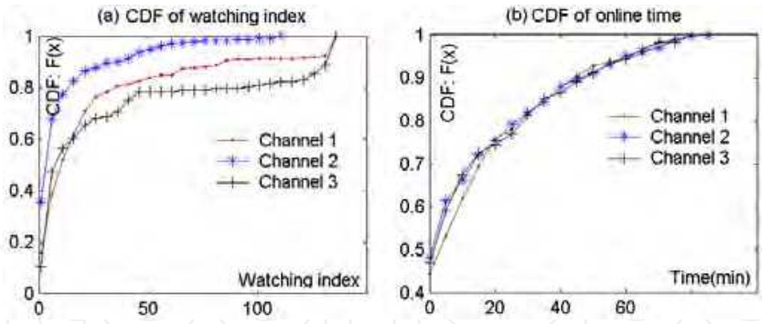

Fig. 13. CDF of WI and online time of contributors

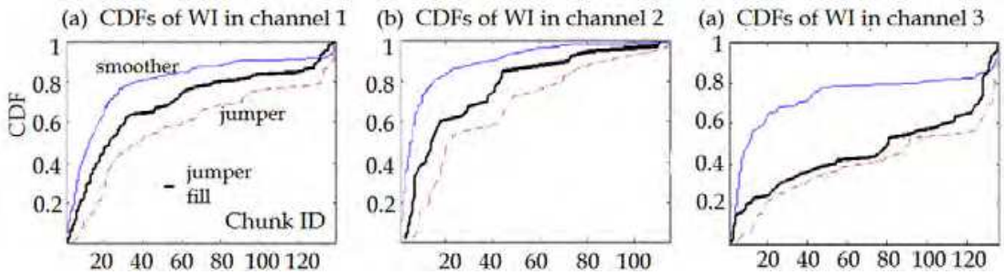

Fig. 14. BM occupancies of contributors

4.2.2 User behavior understanding in terms of watching index

WI help us in better understanding user behavior. Fig.14 shows the CDF of WI for smoothers and jumpers in contributors. The x-axis of each subfigure is the chunk ID or bit positions in the BM. The y-axis is the fraction of the peers. The top curve and bottom curve are of smoother and jumper respectively. The middle curve is the fraction of jumper who has value 1s in its BM at a given bit position. As a peer frequently advertise its BM to others, those subfigures can also be interpreted as the sharing map among VoD peers. Based on this interpretation, we can draw the following conclusions: *i).* although most users are smoother, it may not be good for file-sharing. As lots of people only watch a few chunks, it may lead to overprovision around the initial chunks while under provision for the rest chunks; *ii).* Jumper promotes file-sharing. In each subfigure, the middle curve is significantly below the top curve line. It indicates a jumper contributes more chunks than a smoother. Furthermore, the bottom curve indicates jumpers contribute those chunks with large IDs which smoothers are incapable of sharing; *iii).* Even if jumpers contribute fewer chunks as a whole, their existence is still valuable, as the unbalanced provision resulted from smoothers can be compensated to certain degree by jumpers.

5. Model-based analysis of PPLive at system level

In the section we try to discuss the systematic problems and design concerns on performance, scalability and stability. Based on the live peer's startup models, we will analyze PPLive's design goals, and how PPLive tends to reach the goals; VoD network sharing environment will be analyzed and the inherent connection with user behavior will be revealed.

5.1 System stability based on different initial offset placement schemes

We next introduce two initial offset placement schemes either based on the first neighbor's offset lag or based on its buffer width. We will show how different system design goals can be reached under different schemes, and explain why good peer selection mechanism is critical to make the schemes stable.

5.1.1 Initial offset placement based on offset lag

The first model of initial offset placement makes a new peer (called host as before) decide its *initial offset* based on its *first neighbor*'s offset lag (Li & Chen, 2008a). Assume host h chooses a chunk ID θ as the *initial offset* and begins to fetch chunks at time t_0. After a time interval τ_s, the host starts to drain chunks out of buffer to playback. Then, the offset lag of the host is, $L_h=s(t)-f_h(t) =s(t_0)+r(t-t_0) -r(t-t_0-\tau_s)- \theta=s(t_0)+r\tau_s-\theta$.

As a system designer, for minimizing the workload of tracker server, a person hopes that the wanted chunks are fetched as much as possible from other peers instead of tracker. Thus, the initial offset θ should be chosen when at least one BM has been received for a peer p and θ should be appropriate larger than peer p's offset $f_p(t_0)$. On the other hand, too much diversity among offset lags is not good for sharing environment, so a system designer would wish to control the offset lag, i.e., $L_h-L_p = f_p(t_0)+r\tau_s-\theta$.

It seems a good criterion to let the $L_h-L_p=0$. We call such scheme as *fixed padding* (FP) because of $\theta=f_p(t_0)+r\tau_s$ where $r\tau_s$ is a constant padding. However, FP has no design space.

One can easily find that all peers in such a system will have the same *offset lag* $L \leq W_{tk}$. Buffer width is an important design parameter involving playback performance. Larger buffer can improve playback continuity, but does no good to a tracker for consuming more memories. Thus, FP can't fulfill two design goals at same time: large buffer of peer but small buffer of tracker.

Let's consider a more practical scheme named as *proportional placement* (PP) *based on offset lag*, i.e., $\theta = f_p(t_0) + \alpha L_p$, where α is constant placement coefficient less than 1, and L_p is the first neighbor's offset lag. Since the first neighbor must have been a new peer when it entered the system, we can refer to a very familiar formula $x_{n+1} = bx_n + c$, which is a contraction mapping when the Lipschitz condition satisfies $b < 1$. One can easily concludes that such a system has a stable point $L^* = r\tau_s / \alpha$, which is independent of any specific initial offset.

Self-stabilizing is the most attractive property of *proportional placement* scheme. However, in certain extreme conditions, it may lead to a poor performance. For example, the first neighbor has an offset lag of L_p=1000 but only contains 50 chunks in his buffer. With a placement coefficient α=0.3, the host's $\theta_h = f_p(t_0)$+300, and the host doesn't have any available chunk for download.

5.1.2 Initial offset placement based on buffer width

Instead of offset lag, a host can use $W_p(t)$ for its initial offset placement, where peer p is its first neighbor. We name such a placement scheme as the PP scheme based on buffer width, i.e., $\theta = f_p(t_0) + \alpha W_p(t_0)$. The advantage of this scheme is that, the initial chunk is always available in its neighbor peer. However, the system under this scheme may be not always stable, i.e., this scheme can't guarantee a bounded *offset lag* $L_n = s(t) - f_n(t)$ as $n \to \infty$. In theory, lemmas 1,2 and 3 in (Li & Chen, 2008a) give the offset lag's variant boundaries under certain assumed conditions in line with real situation. According to the lemmas, the measured $E(W)/r$=208.3, and the measured *placement coefficient* α=0.34, then we can deduce the *offset setup time* τ_s=70.82s. The deduced τ_s is very close our measurement result. Hence, the placement scheme used in PPLive is stable.

5.2 The system design concerns based on the TB piecewise line model

Recall the normalized piecewise line design model (Li & Chen, 2008b) of peer startup progress in PPLive. Assuming each stable peer has a offset curve $f(t) = t$ and scope curve $\xi(t) = s(t) = t + W^*$,when peer p arrives at time 0, he has to choose an *initial offset* θ_p relative to a *neighbor's offset* equals as $\theta_p = \tau_s$, which has been confirmed in previous sections as both of them equal 70s. Besides, because the stable peer's buffer width W^* is 210s, thus we see that θ_p is just equal to $W^*/3$.

Offset setup time τ_s is roughly the startup latency and the buffer width W^* is the playback lag to the seeder. Usually, people would like to use large buffers to ensure playback continuity. In our model, θ_p is totally decided by τ_s. So why do not people choose a smaller τ_s for shorter startup latency? Smaller τ_s leads to smaller θ_p. A starting peer must ensure to download θ_p within time τ_s, otherwise, it will miss out it. Thus smaller τ_s means larger download rate γ_p requirement. In fact, for a given τ_s, the minimal γ_p required for fetching all initial B chunks (chunks ID from θ_p to $\theta_p + B - 1$) is about $r_{min} = B/(\tau_s + B)$ since no one chunk can

survivor after τ_s+B seconds. if one wants to decrease the τ_s from 70s to 35s, the peer needs a faster r_{min} , which will impact the startup performance.

There is a physical explanation for the turnover threshold $c_{sch}(t)$ in PPLive. We have $c_{sch}(t)=t+\beta$ *for* $t\geq\tau_s$, which happens to be *the seeder offset* $f_{tk}(t)$ (reported by tracker) since $f_{tk}(t)=t+W^*-120=t+90=t+\beta$. Intuitionally, chunks below $f_{tk}(t)$ may can only be buffered by peers, but those above $f_{tk}(t)$ are cached by both seeder and peers. Designers would like to see a system where peer caches as many as useful chunks while seeder doesn't, and make use of stable peers' resources to help startup peers. Thus β has a lower limit as $\beta\geq 90$.

On the other side, let $v_q(t)=V_q(t)+t$ be the playable video of a neighbor peer q. All chunks below $v_q(t)$ are already fetched by peer q at time t, but chunk $v_q(t)+1$ definitely is not. If we set $c_{sch}(t)>v_q(t)$ and assume q is the only neighbor for host p, then after peer p gets chunk $v_q(t)$, he has to idly wait for peer q to fetch chunk $v_q(t)+1$ according to the TB protocol. If we design $c_{sch}(t)<v_q(t)$, peer p will not waste its time. Substituting model parameters into it, we have $\beta<V_q(t)+t-\tau_s$, for $0\leq t<\tau_s$. If any possible download rates are considered, the right side of the inequality has a minimal value $V_q(t)-\tau_s$. If further assuming $V_q(t)$ for any peer q has the same distribution with a mean V^* and stand deviation σ_V, then we deduced another design rule (Li & Chen, 2010) for the upper limit of β as $\beta<V^*-\alpha\sigma_V-\tau_s$ for $0\leq t<\tau_s$, where coefficient α is introduced to guarantee the switch threshold is below the playable video of his neighbor with larger probability. Based on our measurement in PPLive, V^* is about 196 and σ_V is about 18. For a threshold of 90, α is 2. Through the discussion of system design considerations, we hope to support the claim that PPLive is a well-engineered system.

5.3 VoD network sharing environment

In P2P-based file sharing systems, the number of copies is an important indication to the availability of data blocks. We define the number of copies of chunk θ in the network at a given time t as *availability* $N(\theta,t)$, which equals the number of online peers having this chunk at this time. Our statistics shows that chunks with larger IDs have less availability. Moreover, if we normalize $N(\theta,t)$ by the total number of online peers $N(t)$, or the total number of copies $C(t)=\sum_\theta N(\theta,t)$ at time t, then both the results of $\eta(\theta,t)=N(\theta,t)/N(t)\approx\eta(\theta)$ and $\xi(\theta,t)=N(\theta,t)/C(t)\approx\xi(\theta)$, can be observed independent of time t. We named these normalized availabilities as the *sharing profile* (SP) $\eta(\theta)$, and *sharing distribution* (SD) $\xi(\theta)$. $\xi(\theta)$ is a probability distribution as $\sum_\theta \xi(\theta)=1$, while $\eta(\theta)$ is not. Both SP and SD are shown in Fig.15. In each subfigure there are 86 curves in light color, which correspond to $\eta(\theta,t)$ or $\xi(\theta,t)$ calculated at 1, 2, . . . ,86 minutes of our trace time. Clearly, all the light color curves in each subfigure are very similar. This indicates that the SP and SD are well defined in a practical P2P VoD system.

The user watching behavior will affect the network sharing environment, and an inherent connection does exist between user behavior and VoD network sharing, i.e. the SP and SD can be analytically deduced from the distribution of WI. The theorems 1 and 2 in (Li & Chen, 2010) further verify the time-invariant property of SP and SD. yIn Fig.15, the thick black curve is the result theorem 1 in (Li & Chen, 2010). Clearly, the thick curves match the measured light color curves quite well in all subfigures. Equation 3 in theorem 1 says that the average number of copies is related to the second moment of WI. It indicates that the diversity in users' behaviors can promote network sharing, and this provides twofold

insights: *i).* the system design should facilitate any kinds of viewing habits, such as watching from the middle, watching by skipping and even watching backward. *ii).* a movie should be designed to lure its viewers to present different behaviors, e.g., guiding viewers go to different sections by properly designed previews. In addition, the network sharing research based on the jumpers' WI has a similar result. In short, a good VoD system should be well-designed on both protocols and contents to accommodate any kind of audience.

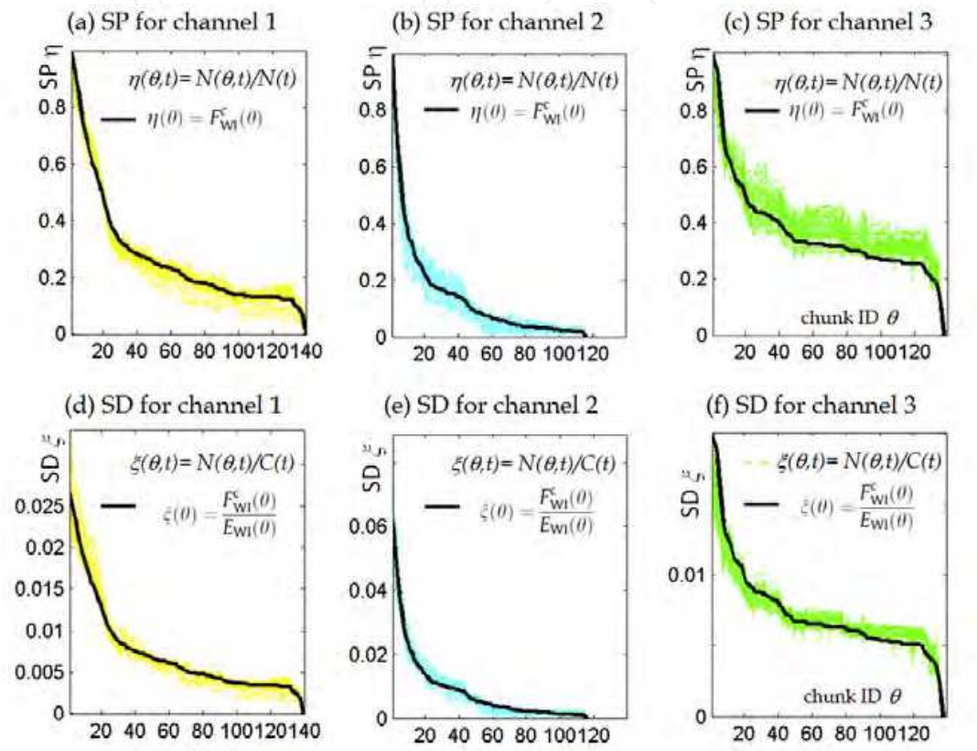

Fig. 15. Sharing profiles (SP) and sharing distributions (SD) in three channels

6. Conclusion

In this chapter, we presented the study of a P2P streaming media system at different levels of detail. The aim of the study is to illustrate different types of analyses and measurements which can be correlated to reverse-engineer, and further give guidelines to optimizing the behavior of such a system in practice. On signaling message level, we tell about our system crack procedure and reveal the protocol flow and message format. Following that, large-scale measurements are carried out with our network crawlers and mass raw data is captured. On peer behavior level, the startup process of live peer is analyzed in two aspects including initial offset placement and chunk fetch strategies. We discover that the initial offset is the only decision factor to a peer's offset lag (playback delay), and the initial offset selection follows certain proportional placement models based on first neighbor's buffer width or offset lag in PPLive. Once the initial offset is determined, a peer downloads wanted chunks following a TB protocol, which can be depicted by a model of a bunch of piecewise lines. Our measurement proofs that in PPLive most live peers (more than 90%) have seemingly good performance. Moreover, VoD peer's behavior is discussed in user (person)

behavior. With the help of measurable parameter of WI, we reveal that although majority peers are smoothers, jumpers tend to be the real valuable file-sharing contributor. On system level, the systematic problems and design concerns on performance, scalability and stability are discussed. Based on the live peer's startup models (PP models and piecewise line model of TB protocol) driven by our trace, we analyze P2P live system's design goals such as the large buffer in peer/small buffer in seeder and self-stability on offset lags, and confirm PPLive tends to really reach those goals. VoD network sharing environment is analyzed in terms of network sharing profile and sharing distribution, and we find the sharing environment is heavily affected by user viewing behavior.

In addition, we will further our study on following issues. We believe live peer chooses its initial offset base on good neighbour, but the evaluation principle of good peer is not answered; The playback rate change has been found in a system designed for CBR video. It needs to be analyzed whether the system can keep in good health when playing a VBR video and how to improve the performance. Besides, we will continue to study what have changed in the continually updated systems.

7. References

Ali, S.; Mathur, A. & Zhang, H. (2006). Measurement of commercial peer-to-peer live video streaming, *In Proceedings of ICST Workshop on Recent Advances in Peer-to-Peer Streaming (2006),* Aug. 2006.

Cheng, B.; Liu, X.Z.; Zhang, Z.Y. & Jin, H. (2007). A measurement study of a peer-to-peer video-on-demand system, *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS'07),* Washington, USA, Feb. 26-27, 2007

Hei, X.; Liang, C.; Liang,J.; Liu Y. & Ross ,K.W. (2007a). A measurement study of a large-scale P2P IPTV system, *Journal of IEEE Transactions on Multimedia*, Oct. 2007, Volume 9, Issue 8, (Dec. 2007), pp. 1672-1687, ISSN 1520-9210

Hei, X.J.; Liu, Y. & Ross, K. (2007b). Inferring Network-Wide Quality in P2P Live Streaming Systems, *IEEE Journal on Selected Areas in Communications*, Vol. 25, No. 10, (Dec. 2007), pp. 1640-1654, ISSN : 0733-8716

Huang, C.; Li, J. & Ross, K.W. (2007). Can internet video-on-demand be profitable? *Proceedings of ACM Sigcomm 2007*. pp. 133-144, ISBN 978-1-59593-713-1, Kyoto, Japan, 27-31 Aug., 2007

Li, C.X. & Chen C.J. (2008b). Fetching Strategy in the Startup Stage of P2P Live Streaming Available from http://arxiv.org/ftp/arxiv/papers/0810/0810.2134.pdf

Li, C.X. & Chen C.J. (2008a). Initial Offset Placement in P2P Live Streaming Systems http://arxiv.org/ftp/arxiv/papers/0810/0810.2063.pdf

Li, C.X. & Chen C.J. (2009). Inferring Playback Rate and Rate Resets of P2P Video Streaming Transmissions by Piecewise Line Envelope Approximation. *Journal of China Univ. of Post and Telecom.*, Vol.16, Issue 2, (April 2009), pp. 58-61, ISSN 1005-8885

Li, C.X. & Chen C.J. (2010). Measurement-based study on the relation between users' watching behavior and network sharing in P2P VoD systems. *Journal of Computer Networks*, Vol.54, Issue 1, (Jan. 2010), pp. 13-27, ISSN 1389-1286

Lou, D.F.; Mao, Y.Y. & Yeap T.H. (2007). The production of peer-to-peer video-streaming networks, *Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, pp. 346-351,ISBN 978-1-59593-789-6, Kyoto, Japan, 31 Aug. 2007

Small, T.; Liang, B. & Li, B. (2006). Scaling laws and tradeoffs of peer-to-peer live multimedia streaming. Proceedings of ACM Multimedia 2006, pp.539-548, ISBN 1595934472, Santa Barbara, CA, USA, 23-27 Oct. 2006

Tu, Y.C.; Sun, J.Z.; Hefeeda, M. & Prabhakar, S. (2005). An analytical study of peer-to-peer media streaming systems, *Journal of ACM Transactions on Multimedia Computing, Communications, and Applications,* Vol.1, Issue 4, (Nov. 2005), ISSN 1551-6857

Vlavianos,A.; Iliofotou,M.; Faloutsos,M. & Faloutsos, M. (2006). BiToS: Enhancing BitTorrent for Supporting Streaming Applications, *Proceedings of INFOCOM 2006*, pp.1-6, ISSN 0743-166X, Barcelona, 23-29 April 2006

Vu, L.; Gupta, I.; Liang, J. & Nahrstedt, K. (2006). Mapping the PPLive Network: Studying the Impacts of Media Streaming on P2P Overlays, *UIUC Tech report*, August 2006

Xu, D.Y.; Chai, H.K.; Rosenberg, C. & Kulkarni, S. (2003). Analysis of a Hybrid Architecture for Cost-Effective Streaming Media Distribution, *Proceedings SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN'03)*, San Jose, CA, Jan. 2003.

Yu, H.L.; Zheng, D.D.; Zhao, B.Y. & Zheng, W.M. (2006). Understanding user behavior in large-scale video-on-demand systems, *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (Eurosys'06)*, pp. 333-344, ISBN 1-59593-322-0 Leuven,Belgium, April 18-21, 2006.

Zhang, L.; Liu, Z. & Xia, C.H. (2002). Clock Synchronization Algorithms for Network Measurements, *Proceedings of the IEEE INFOCOM 2002*, Vol.1, pp. 160-169, ISBN 0-7803-7477-0, New York, USA, June 23-27, 2002,

Zhang, X.;Liu, J.;Li, B. & Yum, T.-S.P. (2005). Coolstreaming/donet: A data-driven overlay network for Peer-to-Peer live media streaming,. *Proceedings of INFOCOM 2005*, vol. 3, pp.2102-2111, ISBN 0743-166X, Miami, FL, USA, 13-17 March 2005

Zheng, C.X.; Shen, G.B. & Li, S.P. (2005). Distributed pre-fetching scheme for random seek support in peer-to-peer streaming applications, *Proceedings of the ACM Workshop on Advances in Peer-to-Peer Multimedia Streaming (P2PMMS'05)*, pp. 29-38, ISBN 1-59593-248-8, Singapore, Nov. 11, 2005

Zhou,Y.P.; Chiu, D.M. & Lui, J.C.S. (2007). A Simple Model for Analyzing P2P Streaming Protocols, *Proceedings of international conference on network procotols 2007*,pp.226–235, ISBN 978-1-4244-1588-5, Beijing, China,5 Nov. 2007

Reverse Engineering - Recent Advances and Applications
Edited by Dr. A.C. Telea

ISBN 978-953-51-0158-1
Hard cover, 276 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Reverse engineering encompasses a wide spectrum of activities aimed at extracting information on the function, structure, and behavior of man-made or natural artifacts. Increases in data sources, processing power, and improved data mining and processing algorithms have opened new fields of application for reverse engineering. In this book, we present twelve applications of reverse engineering in the software engineering, shape engineering, and medical and life sciences application domains. The book can serve as a guideline to practitioners in the above fields to the state-of-the-art in reverse engineering techniques, tools, and use-cases, as well as an overview of open challenges for reverse engineering researchers.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen