Peer-assisted Video Streaming with RTMFP Flash Player: A Measurement Study on PPTV

Shan Zou, *Student Member, IEEE,* Qiang Wang, Junqiang Ge, and Ye Tian, *Member, IEEE*

***Abstract*—RTMFP is a protocol developed by Adobe for multimedia delivery under both client-server and peer-to-peer paradigms. Currently, major Internet video service providers such as PPTV and iQIYI have already built their web-based video streaming systems with RTMFP. In such a system, user only needs to install a Flash Player plug-in on his web browser, and can stream videos in a peer-assisted way.**

Despite its wide usage, RTMFP has received little attention from the measurement community. In this paper, we select PPTV as an example, and study the RTMFP video streaming technology with a measurement approach. We reveal the architecture of PPTV's RTMFP streaming system, and show that comparing with proprietary P2P networks, the RTMFP network has a different content distribution policy, and exhibits different features on peers' streaming behaviors, potential system bottleneck, and network dynamics. We also study RTMFP's video transmission, and find that the protocol's selective retransmission scheme can effectively overcome packet losses and improve the video playback quality; however, the TCP-like congestion control mechanism of RTMFP does not lead to fairness between RTMFP and TCP traffics, due to the mismatch between the inherited pull-based video segment distribution model of the P2P streaming application and the protocol's built-in congestion control mechanism. Our work provides insights on the RTMFP-based video streaming technology, and is helpful for people to construct better peer-assist video systems with RTMFP.

***Index Terms*—Peer-assisted video streaming, RTMFP protocol, network measurement**

I. Introduction

Over the last decade, video has become more and more prevalent on web. It is reported that online video constituted 50% of the web traffic in 2013 [1], and Cisco estimates that by the year 2018, 84% of the web traffic will be video [2].

To enable user to stream videos on web, Internet video service providers such as YouTube [3] and Netflix [4] adopt a browser-server (B/S) architecture, where user employs Sliverlight or Flash Player browser plug-in to stream videos from dedicated CDN servers through HTTP or DASH [5]. In addition to the B/S architecture, many providers seek to enable peer-assisted video streaming on web. The conventional way is to develop a browser plug-in that integrates the provider's proprietary P2P streaming protocol, and ask user to install the plug-in before streaming. Examples of such systems include CNTV and Xunlei Kankan [6]. A problem of this approach is that users may not be willing to install many different proprietary plug-ins from different providers.

TABLE I
Architectures and streaming protocols of popular video service providers in China

Service provider	Architecture	Streaming protocol
Youku	CDN	HTTP
CNTV	P2P	UDP based proprietary protocol
Tudou	CDN	HTTP
PPTV	CDN + P2P	HTTP + RTMFP
iQIYI	CDN + P2P	HTTP + RTMFP
Sohu TV	CDN + P2P	HTTP + RTMFP

Since 2008, Adobe starts to develop the Secure Real-Time Media Flow Protocol (RTMFP) [7], which delivers multimedia content under both browser-server and peer-to-peer (P2P) paradigms, in Flash Player and its other products such as Adobe Integrated Runtime (AIR) and Adobe Media Server. The benefit of RTMFP over provider's proprietary protocol is obvious, as a user only needs to install the Flash Player plug-in, which already exists on 98% of the PCs that connect to the Internet [8], and can stream videos from various service providers as long as these providers build their streaming systems with RTMFP. Nowadays RTMFP has been widely employed by many major video service providers, as we can see in Table I, which lists the architecture and streaming protocol design choices of the six most popular video sites in China, that providers like PPTV[1], iQIYI, and Sohu TV all use RTMFP to provide their online video streaming services on a large scale.

Despite its wide usage, RTMFP protocol and RTMFP-based video systems have received relatively little attention from the measurement community. In this work we select PPTV, which provides web-based live and on-demand video streaming services with RTMFP, as an example, and study how RTMFP enables peer-assisted video streaming with a measurement approach. Our objectives in this paper are twofold:

- **A system-wide measurement study on PPTV's RTMFP-based video streaming system**. We seek to uncover the architecture of PPTV's RTMFP video streaming system, analyze the merits and pitfalls of the RTMFP networks that deliver live and on-demand video contents.
- **A packet-level measurement study on the RTMFP protocol**. We seek to investigate the data transmission of the RTMFP protocol under the context of peer-assisted

The authors are with the Anhui Key Lab on High Performance Computing and Applications, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, P. R. China. Phone: +86-551-63602445, Fax: +86-551-63601013, E-mail: {jelly33, wqiang09, gjq1993}@mail.ustc.edu.cn, yetian@ustc.edu.cn

Corresponding author: Ye Tian (yetian@ustc.edu.cn)

[1]Formerly known as PPLive.

video streaming, and study how the protocol handles packet losses and network congestions.

Unlike the previous measurement studies that focus on proprietary P2P streaming systems with a black-box approach (*e.g.*, [9] and [10]), in this paper, we consider the general-purpose RTMFP protocol and present a gray-box methodology by reverse-engineering Adobe Flash Player. We focus on both the system and the protocol-level issues of the RTMFP-based streaming technology, and make the following contributions in this paper:.

- We present a methodology for studying RTMFP networks by reverse-engineering Adobe Flash Player with a hook DLL, and develop a measurement testbed for investigating RTMFP video streaming systems.
- We uncover the architecture of PPTV's RTMFP-based video streaming system and analyze the system's behaviors and performances. We find that the RTMFP network for live video streaming works as an auxiliary network for reducing the CDN's workload; there is considerable overhead on the centralized RTMFP introduction service, making it a potential system bottleneck; and the RTMFP network is more dynamic than its proprietary P2P network counterpart. We also find that RTMFP suffices to support a VoD service, but CDN has a significant meaning in reducing the seeking delays.
- We investigate the video transmission of the RTMFP protocol, and find that RTMFP's selective retransmission scheme is effective to overcome packet losses and improve the playback quality under lossy network conditions. However, the TCP-like congestion control mechanism of RTMFP does not lead to fairness between RTMFP and TCP traffics, due to the mismatch between the inherited pull-based video segment distribution model of the P2P streaming application and RTMFP's built-in congestion control mechanism.

Note that we are not claiming our study presents the general behaviors of RTMFP video streaming systems, as RTMFP only provides the server-player and player-player communication channels, while a video service provider is free to implement its policies within the server-side and player-side applications upon the protocol stack. However, as the first measurement study on RTMFP protocol and PPTV, a large RTMFP-based video streaming system in real world, we provide insights that are valuable for people to construct better peer-assist video streaming systems with RTMFP.

The remainder part of this paper is organized as follows. In Section II, we briefly introduce the RTMFP protocol and RTMFP networking systems in general. We describe our methodology for measuring RTMFP networks in Section III. In Section IV, we investigate PPTV's RTMFP-based video streaming system. We present our study on RTMFP's video transmission in Section V. Section VI discusses the related work and we conclude this paper in Section VII.

II. Overview of RTMFP Networking

A. A Brief Introduction on RTMFP Protocol

The Secure Real-Time Media Flow Protocol (RTMFP) [7] is a protocol suite developed by Adobe for enabling multimedia content delivery under both client-server and peer-to-peer paradigms. RTMFP was originally a proprietary protocol, and was later opened up and published as IETF RFC 7016[2] in 2013. In this section, we briefly introduce the key concepts and operations in RTMFP.

1) Session: Unlike the TCP-based RTMP protocol [11], RTMFP packets are carried in UDP. An RTMFP packet contains zero or multiple *chunks*, which is the smallest data unit in the protocol.

In RTMFP, two endpoints (*e.g.*, Flash Player instance or RTMFP-capable server) communicate to each other within a session. To set up a session, one endpoint, say endpoint A, initializes a handshake by sending an *IHello* chunk to the other endpoint B; on receiving IHello, if B agrees the session, it returns an *RHello* chunk; then A and B each compute half of the session key and exchange the *IIKeying* and *RIKeying* chunks to complete the handshake. After establishing the session, the two endpoints can setup one or multiple flows and use the *userData* and *Ack* chunks to transmit the user data. An endpoint can also send a *Ping* chunk to elicit a *PingReply* chunk from the other party, so as to check liveness of the session and estimate the latency.

In addition to ordinary endpoint, RTMFP also defines a special endpoint called *redirector*. On receiving an IHello chunk, instead of replying with RHello, a redirector returns with a *Redirect* chunk that contains a list of candidate endpoints with their addresses, and the IHello-sending endpoint selects one or multiple candidates from the list and initializes sessions with them.

In RTMFP, all chunks are encrypted when transmitted on the Internet. The chunks for session handshake are encrypted with the default session key specified by the application specific Cryptography Profile, while after the session is established, chunks such as userData and Ack are encrypted with the per-session key that is negotiated by the two endpoints during the handshake.

2) Flow: An RTMFP session may contain zero or multiple unidirectional *flows*. A flow can be viewed as a sequence of userData and Ack chunks transmitted between a sender and a receiver. For enabling a reliable and orderly chunk transmission, each userData chunk in an RTMFP flow is assigned with a continuously increasing *sequence number* (*SN*). RTMFP defines two types of Ack chunks, namely *rangeAck* and *bitmapAck*, and a receiver can use either of them to acknowledge the userData chunks that it has received and to indicate the lost chunks explicitly.

In RTMFP, a sender does not necessarily resend all the lost chunks, but employs a *selective retransmission scheme*. More specifically, for each flow the sender maintains a variable named *forward sequence number* (*FSN*), and only sends (or resends) any chunk with its SN larger than FSN in the flow. To

[2]However, the RFC is not an IETF Internet Standards Track specification, but for information purpose only.

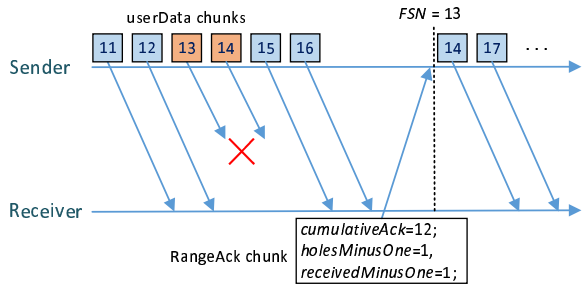


Fig. 1. Demonstration of selectively retransmitting lost RTMFP userData chunks.

inform the receiver of FSN, the sender specifies a field in its userData chunk named *fsnOffset*, whose value is the difference between the chunk's SN and the current FSN value. An example of RTMFP's selective retransmission is demonstrated in Fig. 1: when the sender receives a rangeAck chunk from the receiver with "$cumulativeAck = 12$; $holesMinusOne = 1$, $receivedMinusOne = 1$" in its payload, it infers that a "hole" of $holesMinusOne + 1 = 1 + 1 = 2$ userData chunks with their SNs starting from 13 are lost, and $receivedMinusOne + 1 = 1 + 1 = 2$ userData chunks after the hole with their SNs starting from 15 are properly received; however, since the sender's FSN value is 13 at the moment that the rangeAck is received, it only retransmits chunk 14, and "move on" with new userData chunks. Obviously, such a selective retransmission makes RTMFP very suitable for applications that require timely content delivery.

All the flows in a same RTMFP session share a same congestion control status. RFC 7016 states that RTMFP's congestion control is implemented in a *TCP-compatible* way. That is, an RTMFP sender maintains a transmission budget, which is the amount of the user data that is allowed to be outstanding, or in flight, and the sender follows the congestion control and avoidance algorithms that are in accordance with RFC 5681 [12] to increase and decrease its transmission budget in the flow.

B. Networking RTMFP Flash Player Instances

A typical RTMFP networking system is composed of a number of Flash Player instances and at least one RTMFP-capable server that is connected by these players. The servers provide services such as rendezvous and peer introduction, but they do not get involved in the content distribution. With the server's assistance, a player maintains and updates its neighbors, and exchanges multimedia data by directly connecting to them with RTMFP channels [13].

The RTMFP protocol stack in Flash Player and RTMFP-capable server only provides server-player and player-player communication channels. To provide an Internet service with RTMFP, the service provider need to develop its own server-side and player-side applications. A developer can use Adobe's ActionScript 3.0 programming language [14] to develop the player-side application. The ActionScript code is usually compiled as a shock wave flash (SWF) file and runs on Flash Player instances. For the server-side application, it can be

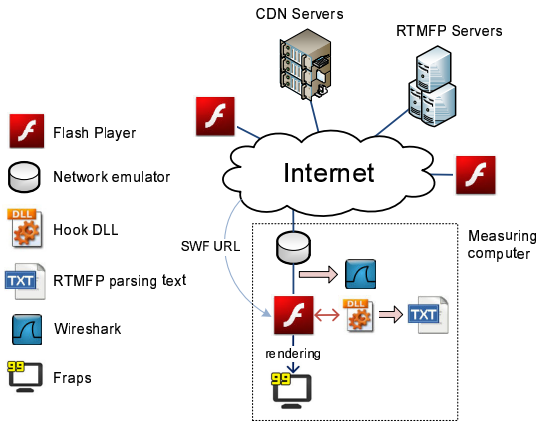


Fig. 2. Measurement testbed.

developed with Adobe's server-side ActionScript language, or be developed independently (*e.g.*, the OpenRTMFP project [15]). Note that by developing the server-side and player-side applications, the service provider actually defines its own policies and strategies in running the P2P network.

III. Measurement Methodology

We seek to understand PPTV's RTMFP video streaming system as well as the RTMFP protocol with a measurement study. Although RTMFP is described in RFC 7016, however, we are facing an obstacle that all the RTMFP chunks are encrypted before transmitted on the Internet, thus it is very difficult to identify and analyze the RTMFP traffic. To overcome this problem, we refer an open-sourced Flash Player reverse engineering project [16], and develop a hook dynamic link library (DLL) to parse the RTMFP chunks that are transmitted and received by Flash Player. More specifically, after being injected into a Flash Player instance, the hook DLL dumps all the incoming chunks after the decryption as well as all the outgoing chunks before the encryption, and we refer RFC 7016 to parse the dumped RTMFP chunks and print the results in a plain text log file. Table II lists the parsing results for some RTMFP chunks as example.

With the hook DLL, we carry out the measurement as the following: For a live or on-demand video content, we obtain the URL of the content's SWF file created by the service provider. The SWF file contains the compiled player-side ActionScript code. After opening the SWF file's URL with the hooked Flash Player, the player follows the instructions of the ActionScript code to contact PPTV's RTMFP servers, join and start to stream from the RTMFP network; the injected hook DLL then parses all the incoming and outgoing RTMFP chunks and prints them in plain texts.

We build a measurement testbed around the hooked Flash Player as demonstrated in Fig. 2. For emulating various network conditions, we install *NEWT* [17], a software-based network emulator, on the computer that runs the measurement. NEWT is able to emulate a variety of network attributes, such as random packet loss and available bandwidth, on port, protocol, and direction basis. In addition to parsing RTMFP chunks, we also use *Wireshark* to capture packet-level traces.

TABLE II
EXEMPLARY RTMFP CHUNK PARSING RESULTS

Chunk	Plain text parsing result
IHello	IHello: {$edpType$ =0x0f; $endpointDiscriminator$: '...'; tag: '...'}
RHello	RHello: {$tagLength = 16$; $tagEcho$: '...'; $cookie$: '...'; $cert$: '...'}
Redirect	Redirect: {$tagLength = 16$; $tagEcho$: '...'; $redirectCnt$ = 1; $redirectDst1$: 58.45.9.100:1913}
userData	userData: {$flowID = 2$; $sequenceNumber = 1144$; $fsnOffset = 120$; $Data$: [...]}
rangeAck	rangeAck: {$flowID = 2$; $bufAvail = 261120$; $cumulativeAck = 1528$; $holesMinusOne = 8$, $receivedMinusOne = 98$}

Note that Wireshark captures both RTMFP and non-RTMFP traffics, including the HTTP video streaming traffic from CDN. For evaluating the video playback quality, we install *Fraps* [18], a software that is capable to benchmark the Frames Per Second (FPS) of the video rendered by the Flash Player (in full-screen mode). We run the measurement testbed in our campus network, which has good network connectivities to all the major ISPs in China.

We want to stress that it is not easy to obtain the URL of the SWF file and play it in our hooked Flash Player, as many video service providers take great efforts to protect their SWF files from being located and played on Flash Player instances that aren't embedded in their web pages. Fortunately, after analyzing the complicated interactions between browser and web server, we are able to locate and execute the SWF files from PPTV with our measurement testbed.

IV. MEASURING PPTV'S RTMFP VIDEO STREAMING SYSTEM

In this section, we seek to understand PPTV's RTMFP-based video streaming system, in particular, we are interested in the system's architecture, behaviors of the Flash Player instances as peers, and the policies enforced by the system. Note that in addition to the web-based video streaming, PPTV also runs a proprietary P2P network that allows users to stream videos with the proprietary clients [9][19].

A. System Architecture and Deployment

We analyze the chunk parsing results collected by our measurement testbed to uncover PPTV's RTMFP-based video streaming system. As illustrated in Fig. 3, when loading the SWF file containing the compiled ActionScript code from PPTV's website, the Flash Player first contacts the provider's *RTMFP rendezvous service* at `rtmfp://fppcert.pptv.com:1935` by resolving the name `fppcert.pptv.com` and sending an IHello chunk to the server that binds the name on UDP port 1935 (step 1). On receiving IHello, the rendezvous server responds with a Redirect chunk containing the address of a PPTV's *RTMFP introduction server* (step 2). The player then shakes hands with the introduction server by exchanging IHello, RHello, IIKeying, and RIKeying chunks (step 3). After establishing a session with the introduction server, the player sends an IHello chunk, probably indicating the video information in the tag field (step 4), and on receiving such an IHello chunk, the introduction server returns with a Redirect chunk containing a list of peers that are currently streaming the same video on the RTMFP network (step 5). The player then selects peers

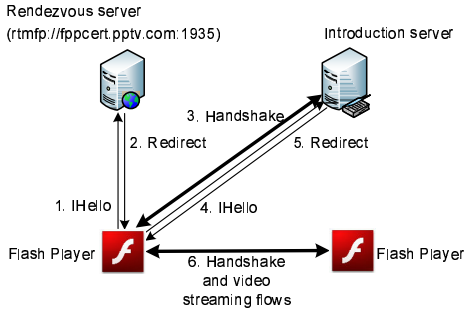


Fig. 3. Demonstration of the major steps in RTMFP video streaming.

from the list, shakes hands with, and streams video data from them (step 6). Note that the player can repeat step 4-6 anytime when it needs to contact more peers.

We employ VPN servers from various ISPs and locations to probe PPTV's RTMFP rendezvous and introduction servers, and find that PPTV places its servers at two locations: one is in China Unicom in Shanghai, and the other is in China Telecom in Hangzhou. We find that the DNS lookup of `fppcert.pptv.com` from Unicom is always resolved to the Shanghai Unicom rendezvous server, while requests from other ISPs are resolved to the Hangzhou Telecom server. However, when answering IHello chunks, the rendezvous server does not necessarily return the introduction server at the same location, but dynamically assign players to Shanghai or Hangzhou servers, probably for load-balancing reason.

Note that in addition to joining the RTMFP network, the Flash Player also contacts the global load-balancing mechanism of PPTV's CDN network, and is directed to appropriate CDN servers for streaming the video at the same time [20].

B. Live Video Streaming with RTMFP

PPTV provides both live and on-demand video streaming services on web with its RTMFP-CDN hybrid system. In this subsection, we first study PPTV's RTMFP network that delivers live video content.

1) Peer streaming behaviors: In our first experiment[3], we launch the hooked Flash Player to stream "Jiangsu TV", a popular TV channel in China, from PPTV's website, and decompose the captured traffic into the HTTP download traffic from the CDN, as well as the RTMFP upload and download traffics.

We present a typical decomposition result of one-hour streaming traffic in Fig. 4. From the figure one can see that

[3]Unless stated otherwise, all the experiments in this paper were conducted between 20:00 and 22:00, and each experiment was repeated at least five times, while we select the most representative result to present.

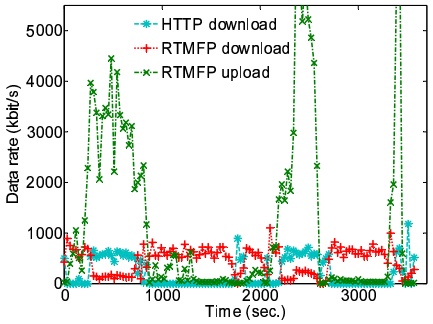


Fig. 4. Traffic decomposition of campus peer in live video streaming.

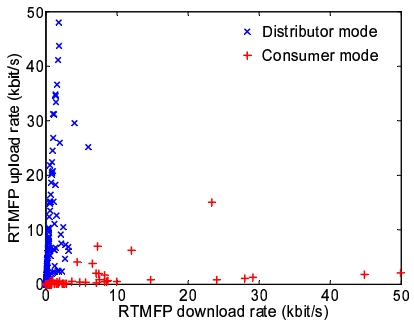


Fig. 5. Upload and download rates of each encountered peer in distributor and consumer modes.

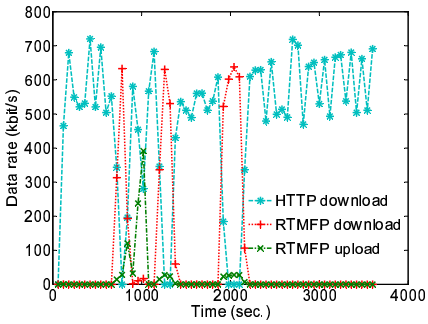


Fig. 6. Traffic decomposition of peer with limited upload bandwidth in live video streaming.

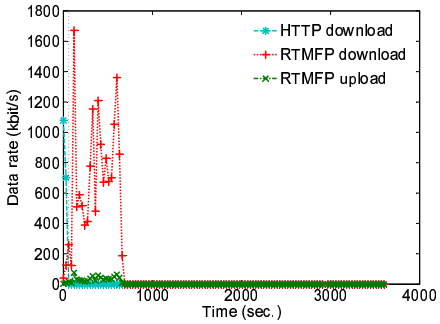


Fig. 7. Traffic decomposition of HTTP-restricted peer in live video streaming.

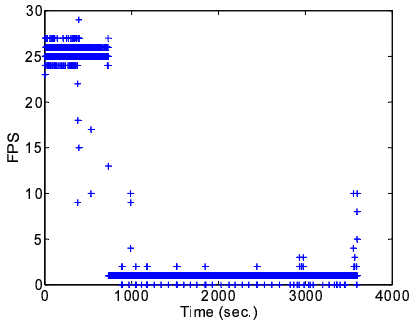


Fig. 8. Live video playback FPS of HTTP-restricted peer.

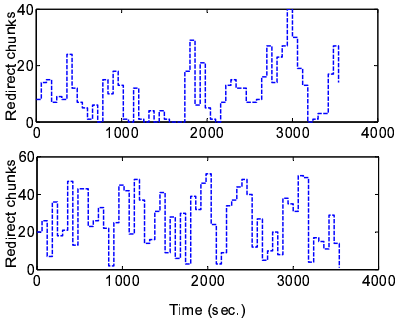


Fig. 9. Redirect chunks received by (top) peer with limited upload bandwidth, and (bottom) HTTP-restricted peer, in live video streaming.

during the intervals of $210 \sim 840$ second, $2,130 \sim 2,580$ second, and $3,330 \sim 3,450$ second, the player streams the video mostly from the CDN, and uploads to peers on the RTMFP network at very high data rates. We refer to these intervals as the *distributor mode* of the player, as during these intervals, the player distributes the video content from the CDN to peers on the RTMFP network. For the other time of the measurement, the player downloads mostly from peers on the RTMFP network, but uploads little to them. We refer to these intervals as the player's *consumer mode*, as the player consumes the content that is distributed by other peers, especially the ones under their distributor mode.

To better understand the behavioral discrepancy of the two modes, for each RTMFP peer encountered by our player, we plot the averaged upload and download data rates between the peer and our player in Fig. 5. In the figure, each plot represents a peer encountered by our player under its distributor or consumer mode, and the plot's x- and y-coordinates are the download and upload data rates respectively. From the figure, we can see that under its distributor and consumer modes, the player uploads and downloads alternately in the RTMFP network, which is very different from the previous observation on proprietary P2P IPTV systems that peers upload and download simultaneously [10].

From Fig. 4, we can see that for a peer to contribute in the RTMFP network, it must meet two conditions: 1) it should have sufficient upload bandwidth; and 2) it should have plenty of video data obtained from the CDN that is "fresh" to other peers on the RTMFP network. In the following experiments, we study the player's streaming behaviors under the cases that either condition is unsatisfied.

We first consider the case that a peer has limited uploading capacity. We employ NEWT to limited the upload bandwidth of our measurement testbed to $800\,\mathrm{kbit/s}$, and use the hooked Flash Player to stream Jiangsu TV. Note that $800\,\mathrm{kbit/s}$ is a typical upload data rate for the ADSL2+ access technology [21]. Fig. 6 presents a representative one-hour streaming traffic decomposition. From the figure we can see that our player makes efforts to stream from its neighbors on the RTMFP network during the three short intervals of $720 \sim 840$ second, $1,200 \sim 1,320$ second, and $1,920 \sim 2,100$ second, but resorts to the CDN for streaming the video eventually. We observe that in all the repeated experiments, the player was always degraded to a CDN-only player.

In the second case, we employ NEWT to restrict the HTTP download data rate to $1\,\mathrm{kbit/s}$ after streaming a TV channel with our hooked player. By restricting the HTTP traffic, the player works under a "pure" P2P mode without being able to stream from the CDN servers. We present a typical traffic decomposition in Fig. 7. From the figure, we can see that the player can still stream from the RTMFP network after being prevented from the CDN for a while. However, as we can see from Fig. 8, which plots the playback FPS captured by Fraps, that after losing its uploading neighbors at the 660^{th} second, the player can no longer stream from the RTMFP network and stops to play.

Our observations on the measurement experiments show that unlike the conventional P2P systems, which seek to

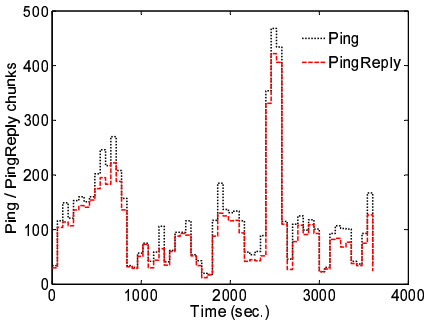


Fig. 10. The Ping and PingReply chunks sent and received by a campus peer in live video streaming.

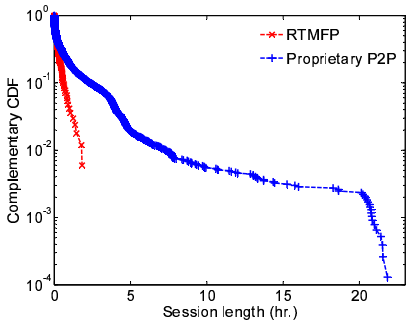


Fig. 11. Distributions of session lengths in PPTV's RTMFP network and proprietary P2P network.

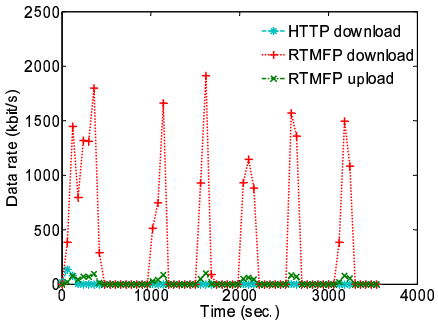


Fig. 12. Traffic decomposition of campus peer in on-demand video streaming.

provide video service to as many peers as possible, for PPTV's web-based live video streaming service, the RTMFP network is not a replacement of the CDN infrastructure, but works as an *auxiliary network* for helping distributing the video content and reducing the CDN's workload. Any peer that fails to contribute to the RTMFP network and mitigate the CDN workload can not benefit from the RTMFP network.

2) Overhead on RTMFP server: As described in Section IV-A, the only way for a peer to renew its neighbor set in PPTV's RTMFP network is to contact the introduction server for new candidates. In the following, we investigate the overhead of the introduction service.

In Fig. 9, we present the numbers of the Redirect chunks that our player has received from PPTV's RTMFP introduction server per minute, for the two measurement experiments that the player has limited upload bandwidth or is prevented from the CDN as in Fig. 6 and Fig. 7 respectively. From the figures we can see that the player contacts the introduction server quite often, as on average, 10.5 and 18.4 Redirect chunks were received by our player per minute. Moreover, by comparing the two figures with Fig. 6 and Fig. 7, we find that when our player stops to stream from the RTMFP network or stops to stream from anywhere, it can still obtain new candidate peers from the introduction sever; and by analyzing the parsing log files, we find that our player is able to complete handshakes and set up sessions with the candidate peers all the time, despite that all its neighboring peers refuse to upload video data to our player.

From the measurement we can see that the overhead on the RTMFP introduction server is non-trivial, and the server indiscriminately provides the service to all the peers. Our observation suggests that for PPTV's RTMFP-based live streaming system, its centralized introduction service could be a potential system bottleneck, and a possible way to alleviate it is to differentiated the peers and only provide service to the ones that actually contribute to the RTMFP network.

3) Network dynamics: A major concern for providing video service with a P2P network is the network dynamics, as a contributing peer may abruptly leave the network anytime. In RTMFP, a player can send the Ping chunks to check the liveness of its neighbors. In Fig. 10, we plot the numbers of the Ping and PingReply chunks that are sent out and received by our player, for the live video streaming experiment in Fig. 4. From the figure we find that among the $7,179$ Ping chunks sent out by our player, as many as $1,058$ do not elicit a PingReply chunk from the other end of the session, the non-reply ratio is close to 15%. The observation suggests that the RTMFP network, which is composed of Flash Player plug-in instances, may be quite dynamic.

To further investigate the dynamics of the RTMFP network, we continuously stream Jiangsu TV for 24 hours with our measurement testbed, and measure the durations of the sessions established between our player and other RTMFP peers. We only consider the sessions with essential video transmission by filtering out those that have less than 10 userData chunks larger than 1500 Bytes. We also run PPTV's proprietary client to stream the same TV channel at the same time, and employ the methodology in [9] to measure the session lengths in PPTV's proprietary P2P network.

We present and compare the distributions of the session lengths in the two networks in Fig. 11. From the figure one can see that the RTMFP sessions with an averaged session length of 14.75 minutes are much shorter than those in the proprietary P2P network, which have a mean session length as long as 46.37 minutes. However, the median session lengths, which are 7.85 minutes for the RTMFP network and 7.39 minutes for the proprietary P2P network, are close to each other.

After carefully examining the sessions, we find that the most significant difference between the two networks are the presence/absence of the long sessions. The longest session we have observed on the RTMFP network lasted less than 2 hours, and such lengths are reasonable for people's TV watching behavior. However, on the proprietary P2P network, we have recorded many sessions that are extraordinary long, for example, there are 56 sessions longer than 8 hours, constituting 0.73% of all the sessions we have collected. Obviously, the absence of the extraordinary long sessions makes the RTMFP network more dynamic than the proprietary P2P network. Our observation also explains the many Redirect chunks as observed in Fig. 9, as a player needs to contact the introduction service frequently to maintain its neighbor set under the highly dynamic network.

We explain the difference on the session lengths between the RTMFP and the proprietary P2P networks as the following. PPTV clients (and most of other proprietary P2P IPTV clients) nowadays run a background service process with the player

TABLE III
EMULATED NETWORK ATTRIBUTES AND FINDINGS ON PPTV'S RTMFP VIDEO SYSTEM.

Emulated network attribute	Findings
None	Player streams live video from CDN and RTMFP network alternately.
Restrict the upload bandwidth to 800 kbit/s	Player no longer streams live video content from the RTMFP network and is degraded to a CDN-only player.
Restrict the HTTP download bandwidth to 1 kbit/s	Player no longer streams live video content from the RTMFP network and stops to play.

by default. After the client's player window is closed, the service process is still running as a peer in the proprietary P2P network and distributes the video data all the tim. These background processes explain the extraordinary long sessions we have observed. However, there is no such a background service process for the RTMFP Flash Player, and the RTMFP session lengths are simply the users' TV channel watching time.

C. On-Demand Video Streaming with RTMFP

We also examine the on-demand video streaming service provided by PPTV with RTMFP. Since unpopular contents are mostly served by CDN, we focus only on popular contents by streaming a popular TV drama in our experiment. Fig. 12 presents a typical traffic decomposition result. From the figure we can see that the player downloads from the CDN only at the beginning of the streaming, while for most of time, it relies on the RTMFP network to receive the service.

We further examine PPTV's RTMFP network for on-demand streaming by restricting the testbed's upload bandwidth and HTTP traffic, and find that PPTV does not enforce a player to contribute to the RTMFP network as in live video streaming. The player simply downloads from all the available sources in an aggressive way.

Although in VoD service, a Flash Player can download most of the video data from the RTMFP network, however, CDN has a significant meaning in reducing the seeking delays, *i.e.*, the delay between the time that a user seeks to a position and the time that the playback starts. To show this, we employ NEWT to prevent and allow the HTTP traffic from the CDN, and measure the seeking delays under the two conditions respectively. We find that when the CDN is prevented, the averaged seeking delay is over one minute, but when the CDN is available, there is nearly no obvious delay after the seeking. The difference is easy to understand, as without the CDN, a seeking player needs to contact the RTMFP introduction server and establish sessions with the new peers before streaming the video data; however, by maintaining a persistent HTTP connection to a CDN server, the player can download immediately after the seeking.

D. Summary

We outline our findings on PPTV's RTMFP video streaming system in Table III and summarize the observations as the following:

- *System architecture*: PPTV's employs an RTMFP-CDN hybrid system to provide live and on-demand video streaming services; the RTMFP network is composed of rendezvous servers, introduction servers, and many Flash Player instances as peers on the RTMFP P2P network.
- *Content distribution policy*: PPTV's RTMFP network for live video streaming works as an auxiliary network for reducing the CDN's workload, where a peer is enforced to contribute to the RTMFP network by streaming from the CDN and from the RTMFP network in the distributor and consumer modes alternately.
- *System bottleneck*: PPTV provides the introduction service in a centralized manner to all the peers indiscriminately, and there are considerable overheads on the introduction servers due to the network dynamics, making the service a potential system bottleneck.
- *Network dynamics*: PPTV's RTMFP network for live video streaming is more dynamic than its proprietary P2P network counterpart, due to the absence of the extraordinary long sessions caused by the background service processes of the proprietary P2P clients.
- *VoD service*: PPTV's RTMFP network for on-demand video streaming service is able to support the service alone; however, CDN has a significant meaning in reducing the seeking delays in the VoD service.

V. MEASURING RTMFP VIDEO TRANSMISSION

In this section, we examine the data transmission of the RTMFP protocol under the context of peer-assisted video streaming. In particular, we focus on how the protocol handles packet losses and congestions. We find that comparing with HTTP/TCP, RTMFP is very suitable for video streaming under a lossy network condition, thanks to its selective retransmission scheme; however, for peer-assisted video streaming, the TCP-like congestion control mechanism does not lead to fairness between RTMFP and TCP traffics, due to the mismatch between the P2P streaming application's inherited pull-based video segment distribution model and RTMFP's built-in congestion control mechanism.

A. Lossy Network and Selective Retransmission

Most user devices nowadays access the Internet with wireless technologies, however, wireless channels including Wi-Fi, 3G, and 4G are inherently unreliable, and many factors, such as transmission distance, competing traffic, relative sender/receiver speed, base-station handoff, etc., can cause high loss probabilities. For example, a measurement study [22] shows that in the classroom environment, a Wi-Fi loss probability as high as 11.5% is observed between a sender and a receiver 40 feet apart; and a recent study [23] reports that on high-speed trains, a TCP receiver over an HSPA+ channel (3.75G) experiences high loss rates all the time.

As a video streaming protocol, RTMFP incorporates a selective retransmission scheme as introduced in Section II to handle chunk losses, that is, an RTMFP sender does not necessarily retransmit all the lost chunks, but only resends the ones with their sequence numbers larger than its FSN.

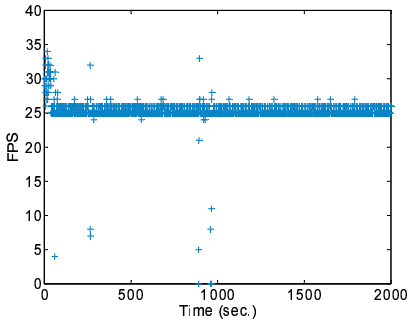


Fig. 13. FPS of RTMFP-only player under the 0.01 lossy downlink.

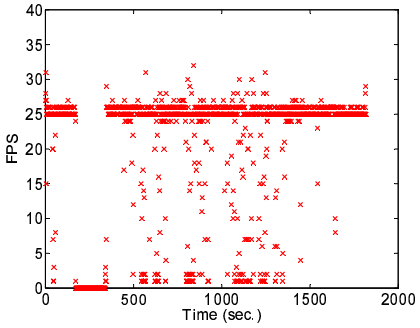


Fig. 14. FPS of CDN-only player under the 0.01 lossy downlink.

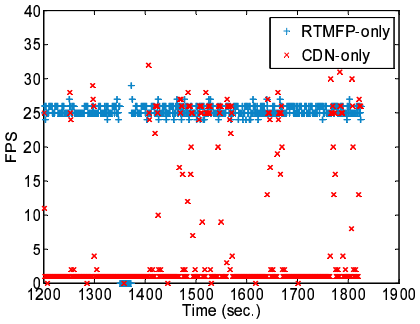


Fig. 15. FPS of RTMFP-only and CDN-only Flash players under the 0.1 lossy downlink.

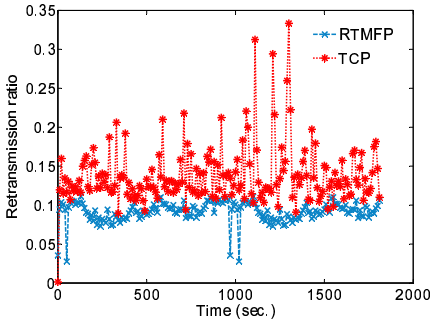


Fig. 16. Retransmission ratios of RTMFP and TCP for live video streaming under the 0.1 lossy downlink.

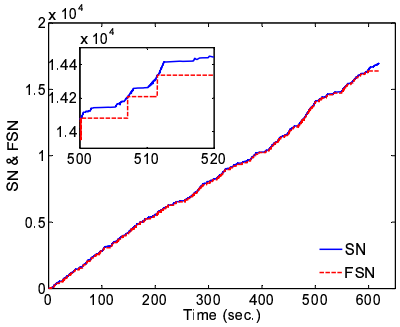


Fig. 17. SN and FSN within the flow from 218.76.58.70:49522 as the sender, under the 0.1 lossy downlink.

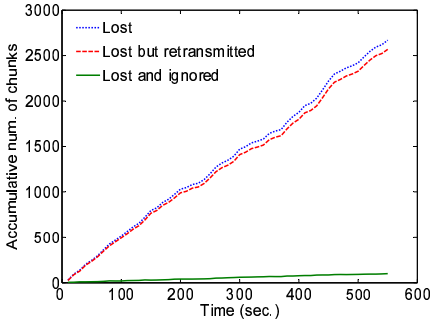


Fig. 18. Acc. num. of lost, lost but retransmitted, lost and ignored userData chunks within the flow under the 0.1 lossy downlink.

TABLE IV
STREAMING SMOOTHNESS OF RTMFP AND HTTP STREAMING UNDER LOSSY DOWNLINKS WITH 0.01 AND 0.1 RANDOM LOSS PROBABILITIES

Loss prob. = 0.01	Buffering incidents	Total buffering time (%)
RTMFP-only	0	0 second (0.0%)
CDN only	27	175 seconds (9.7%)
Loss prob. = 0.1	Buffering incidents	Total buffering time (%)
RTMFP-only	1	4 seconds (0.2%)
CDN only	64	1,405 seconds (78.1%)

In this subsection, we examine the effectiveness of such a selective retransmission scheme for video streaming under lossy network conditions.

In our first measurement experiment, we use the hooked Flash Player to stream Jiangsu TV from PPTV. After 5 minutes of the streaming, we employ NEWT to prevent the HTTP traffic so that the player can only stream from the RTMFP network, and we emulate a lossy downlink with a moderate packet-level loss probability of 0.01. To compare RTMFP with the HTTP streaming over TCP, we perform another experiment, in which we also emulate a 0.01 lossy downlink, but this time we prevent the UDP traffic to force the player to stream from the CDN only. The experiments last over 30 minutes[4], and we use Fraps to capture FPS of the live video played by the Flash Player when it is prevented from the CDN and from the RTMFP network respectively.

[4]Since the player will eventually lose the streaming service after being prevented from the CDN, we repeat the experiment several times until finding a case that the streaming lasted for over 30 minutes.

We present the FPS of the RTMFP-only and the CDN-only players in Fig. 13 and Fig. 14 respectively. From the figures we can see that under the lossy downlink, the CDN-only player can not maintain a smooth video playback at about 25 FPS due to the packet losses, on the other hand, the RTMFP-only player is almost not influenced. We also count all the "buffering" incidents manually and summing up the intervals of these incidents in the first 30 minutes of the streaming. Table IV presents the results. We can see that under the lossy downlink, the CDN-only player has suffered as many as 27 buffering incidents with a total buffering time of 175 seconds; on the other hand, the RTMFP-only player has not experienced any buffering incident at all. Our observation suggests that, as a transmission protocol dedicated to multimedia distribution, RTMFP handles packet losses much better than HTTP/TCP.

As high loss rates are widely observed in wireless networks [22][23], we also emulate a lossy downlink with a packet-level loss probability as high as 0.1, and repeat the experiments. We present the FPS of the RTMFP-only and the CDN-only players in Fig. 15, and summarize all the buffering incidents of the two players in Table IV. We can see that even under a high loss probability, the RTMFP-only player still manages to play smoothly, while the CDN-only player barely plays with 78.1% of the time buffering.

The better performance of RTMFP is easy to understand, as in HTTP streaming, the receiver's TCP protocol stack always waits for the lost segments to be retransmitted by the sender until timeout, but in RTMFP, the receiver can ignore some of the lost chunks as the sender increases its FSN, thus is able to

render more video frames in time. To show this, we calculate the *retransmission ratios* for both protocols at the receiver side in the 0.1 lossy downlink experiment. The retransmission ratio is defined as the ratio between the retransmitted packets our player has received and its total received packets. For TCP-based HTTP streaming, we consider a received TCP segment as retransmitted if the segment's sequence number appears in no less than three duplicated ACKs [12]. While for RTMFP, as introduced in Section II, we first identify all the lost userData chunks by parsing the "holes" in the rangAck chunks that are sent out by our player, then we consider any received userData chunk with its SN falling in a hole of a previous rangeAck chunk as retransmitted.

In Fig. 16, we plot and compare the retransmission ratios of the two protocols in every 10-second interval. From the figure we can see that, for most of the time, the TCP-based HTTP streaming has a higher retransmission ratio than RTMFP, which confirms our analysis that an RTMFP sender selectively retransmits lost chunks while TCP seeks to resend all the lost segments.

To obtain more insights on RTMFP's selective retransmission scheme, we study the incoming flow that transmits most video data in the experiment. The flow is between peer `218.76.58.70:49522` as the sender and our player as the receiver. Fig. 17 plots how the sender's SN and FSN increase over time, and the inset figure shows the SN and FSN in a 20-second interval. From the figure we can see that the Flash Player's RTMFP protocol stack increases a sender's FSN in a very simple way: the FSN remains unchanged as long as the difference between SN and FSN (*i.e.*, fsnOffset) does not exceed 127, but when the difference reaches 127, in the next userData chunk, the sender increases FSN to be equal to SN again.

In Fig. 18, we present the accumulative numbers of the userData chunks that are lost, lost but retransmitted, lost and ignored by the sender in the flow over time. Among the $2,669$ lost chunks, only 100 of them are ignored by the RTMFP sender. The "ignoring ratio" is 3.75%. However, as we have seen in Fig. 15 and Table IV, by ignoring such a small portion of the lost chunks, RTMFP can provide a much better video streaming experience than HTTP under lossy network conditions.

B. TCP-like Congestion Control Reconsidered

It is well recognized that P2P traffics are unfriendly to the traditional TCP-based applications such as FTP and HTTP, and some people believe that if a TCP-compatible congestion control mechanism is enforced on P2P flows, the P2P traffic will be more fair to TCP [7][24][25]. However, we consider that under peer-assisted video streaming, such a belief is in doubt. Taking RTMFP as an example, RTMFP is employed for transmitting video segments, which are small in size, and P2P networks such as PPTV usually apply a BitTorrent-like pull-based model to distribute the segments among the peers dynamically [26]. Consider an unidirectional RTMFP flow. The flow transmits data only when the P2P network schedules the receiver peer to fetch a video segment from the sender peer, but remains idle for the rest of time. Under such a pull-based segment distribution model, an RTMFP flow transmits video segments in an intermittent way, which is very different from TCP flow that transmits bulk data continuously and aggressively (*e.g.*, FTP).

Motivated by the above observation, in this subsection, we seek to examine RTMFP's congestion control mechanism, which according to its specification, follows RFC 5681 [12] and is supposed to be TCP-compatible. Note that in this paper we do not intend to testify whether RTMFP is strictly equivalent to TCP, as there are various versions and mechanisms in behind [27][28][29][30]. We also do not assume that the RTMFP implementation strictly conforms to RFC 7016, although Adobe is both the specifier and the implementer of the protocol. We only make the assumption that the RTMFP protocol stack mimics the general TCP behaviors by maintaining a congestion window, growing and backing off on the events of acknowledgements and losses, and we seek to answer the following question: *under the context of peer-assisted video streaming, will the TCP-like congestion control mechanism of RTMFP as implemented in Flash Player lead to fairness between RTMFP and TCP traffics?*

1) Competing TCP for bandwidth: We first carry out two experiments to evaluate fairness between RTMFP and TCP by investigating how much bandwidth one protocol can steal from the other. In both experiments, we prevent our measurement testbed from PPTV's CDN network, and restrict the testbed's available download bandwidth to $500\,\mathrm{kbit/s}$ with NEWT. For the first experiment, we initially employ the hooked Flash Player to stream a popular TV drama from PPTV's VoD service via RTMFP; we wait until the RTMFP flows grab most of the available bandwidth, then introduce a TCP flow by downloading an Ubuntu Linux image from a server of Baidu's cyberlocker service[5] through HTTP on the measuring computer. We study how much bandwidth the TCP flow can steal from the existing RTMFP flows by comparing the data rates achieved by the two protocols, and present a representative experiment result in Fig. 19. From the figure we can see that the newly arrived TCP flow takes most of the available bandwidth from the existing RTMFP flows.

We then experiment the opposite case of RTMFP stealing bandwidth from TCP. We first download the Ubuntu image from the cyberlocker server and wait for the TCP flow to take all the $500\,\mathrm{kbit/s}$ available bandwidth, then we introduce RTMFP flows by launching the hooked Flash Player to stream a popular TV drama from PPTV's RTMFP network for VoD service. A representative experiment result can be found in Fig. 20. We can see that except for the first few seconds after the introduction of RTMFP, the TCP flow still takes most of the available download bandwidth all the time, while the Flash Player does not have enough bandwidth to stream the video.

Besides the bandwidth stealing, we also carry out an experiment to enable the two protocols to compete for bandwidth directly. In the experiment, we initially stream from PPTV's VoD service and download the Ubuntu image at the same time, when both protocols' traffic become stable, we employ NEWT

[5]http://pan.baidu.com/

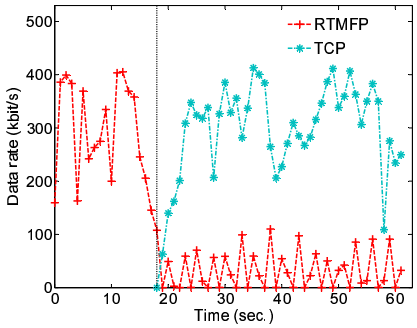


Fig. 19. Bandwidth stolen by TCP from RTMFP.

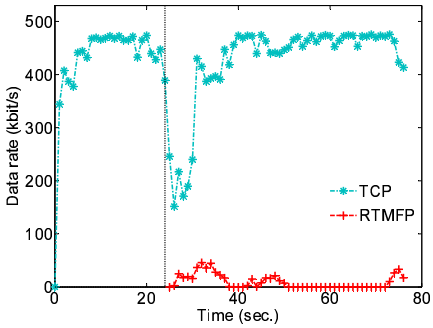


Fig. 20. Bandwidth stolen by RTMFP from TCP.

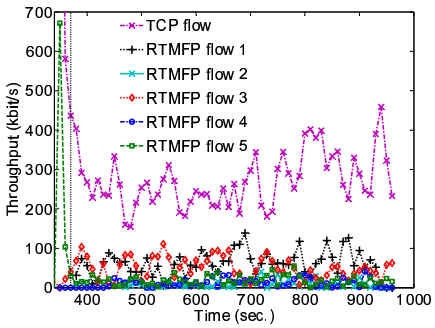


Fig. 21. Throughputs of TCP and RTMFP flows.

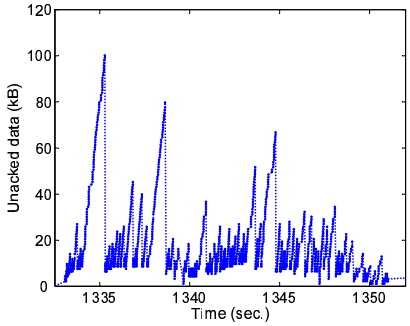


Fig. 22. Unacknowledged data of RTMFP flow during segment transmission.

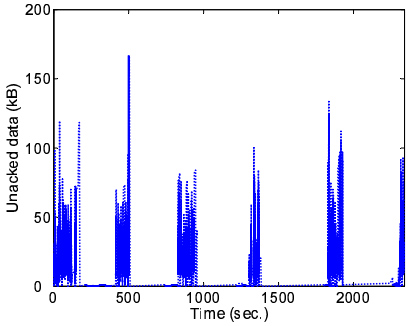


Fig. 23. Unacknowledged data of RTMFP flow during entire measurement.

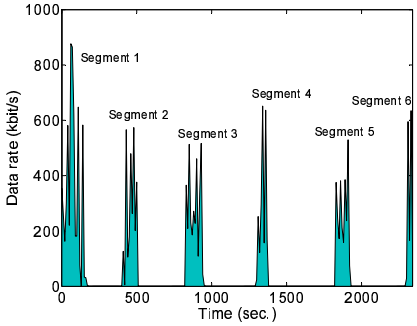


Fig. 24. RTMFP flow throughput that corresponds to video segment transmissions.

to restrict the testbed's download bandwidth to $500\,\mathrm{kbit/s}$, and observe how the two protocols compete. We have identified five RTMFP flows that constitute most of the protocol's traffic, and compare their throughputs with the TCP flow in Fig. 21. One can see that after the bandwidth restriction, the TCP flow achieves a much higher throughput than any individual RTMFP flow all the time.

To filter out the influence of the round trip time (RTT) on the congestion control algorithms, for all the three experiments, we measure the RTTs between our testbed and the cyberlocker server as well as the ones between our player and the five peers that contribute most RTMFP traffics. We estimate that the testbed–server RTT is about $16\,\mathrm{ms}$, while the testbed–peer RTTs vary between $8 \sim 33\,\mathrm{ms}$. The measured RTTs suggest that both the cyberlocker CDN and the RTMFP network schedules nearby sources, and their RTTs are comparable.

Overall, by summarizing the observations from Fig. 19–21, we can see that *under the application context of peer-assisted video streaming, RTMFP is suppressed by TCP when competing for limited bandwidth, despite that RTMFP adopts a TCP-like congestion control mechanism.*

2) Interpreting the TCP aggressiveness: To understand the aggressiveness of TCP over RTMFP, we compute and analyze the unacknowledged (unacked) data in RTMFP flow, as the unacked data can be considered as a good approximation of sender's congestion control window (cwnd) [31]. More specifically, in our experiment, we employ the hooked Flash Player to stream a hot TV drama from PPTV without restricting the testbed's bandwidth, so that the player can upload to other RTMFP peers as fast as possible. We select the most significant uploading flow, and compute its unacked data as sum of the payloads of the continuous userData chunks that are sent out but not yet acknowledged. The unacked data is computed on each event of sending a userData chunk or receiving a rangeAck chunk.

Fig. 22 demonstrates how the unacked data varies over time within a very short interval of about 20 seconds. From the figure we can see that when the player is actively transmitting video data, the unacked data varies in a very similar way as a TCP flow [31]. However, when look at the unacked data over a long period of time as in Fig. 23, which covers the entire measurement, we find that significant unacked data exists in only six intervals. In other words, the RTMFP flow under study only transmits video data actively in the six intervals, but is idle for the rest of time, which constitutes 70% of the entire measurement.

In Fig. 24 we plot the data that is sent out by our player through the RTMFP flow under study. The figure confirms our analysis that the flow only transmits in six intervals, moreover, we find that in each interval, approximately a same amount of video data is transmitted. Fig. 23 and Fig. 24 suggest that as in most peer-assisted video streaming systems, in PPTV's RTMFP network, video is divided into segments and is distributed with a BitTorrent-like pull-based model. In such a model, a peer makes decisions to fetch different video segments from different neighbors, and the decision is made dynamically based on the factors such as playback deadline, segment availability, and neighbors' bandwidths. Note such a pull-based segment distribution model is the de-facto approach in many P2P IPTV systems (*e.g.*, [19], [10], and [32]). From

TABLE V
EMULATED NETWORK ATTRIBUTES AND FINDINGS ON RTMFP DATA TRANSMISSION.

Emulated network attribute	Findings
Downlink of loss probability of 0.1	RTMFP-only player plays smoothly, while CDN-only player loses frames.
Downlink of loss probability of 0.01	RTMFP-only player plays smoothly, while CDN-only player can not play.
Restrict the download bandwidth to $500\,\mathrm{kbit/s}$ and introduce a TCP flow	RTMFP flows are overwhelmed by TCP flow and the player can not play.

Fig. 24, we can estimate that the averaged segment size in PPTV's RTMFP network is about $3.8\,\mathrm{MB}$, which is typical for a P2P VoD service [19].

We explain the aggressiveness of TCP over RTMFP as observed in Fig. 19–21 with the pull-based segment distribution. That is, each time a receiver fetches a video segment from a sender through an RTMFP flow, the flow's congestion window grows from zero, and after the segment transmission is completed, the flow becomes idle and the congestion window drops to zero, until the next time the receiver schedules to fetch a new segment from the sender through the flow again. On the other hand, in a TCP flow for bulk data transmission (*e.g.*, downloading a Linux image), the sender aggressively transmits TCP segments until congestion is encountered, thus the flow can maintain a large congestion window all the time. In our bandwidth competing experiments, each time an RTMFP flow has completed transmitting a segment and becomes idle with a zero-sized congestion window, the flow's bandwidth will be taken by the TCP flow aggressively, therefore it is very difficult for an RTMFP flow or even multiple flows to compete with one single TCP flow, as we have observed in Fig. 19–21.

Note that although the TCP-like congestion control is a built-in mechanism of the RTMFP protocol stack in Flash Player, the pull-based segment distribution model is implemented by PPTV as an application-specific strategy in the player-side SWF file. When combining RTMFP with the video segment distribution model that is designed without considering the transmission protocol's congestion control, as in many UDP-based proprietary P2P streaming systems, we can see that new unfairness arises between TCP and RTMFP traffics. Our observation suggests that to achieve TCP-fairness, a TCP-like congestion control mechanism of RTMFP is not enough, the P2P network's inherited video segment distribution model should also be re-considered carefully.

C. Summary

We outline our findings on the data transmission of the RTMFP protocol in Table V and summarize the observations as the following:

- *Selective retransmission*: RTMFP employs a selective retransmission scheme by selectively ignoring a small portion of the lost chunks without retransmitting them, and such a scheme is very effective to overcome packet losses under lossy network conditions and improve the video playback quality.
- *Congestion control*: Under the application of peer-assisted video streaming, RTMFP's TCP-like congestion control mechanism does not lead to fairness between RTMFP and TCP traffics. The unfairness can be explained with the mismatch between PPTV's inherited pull-based segment distribution model and RTMFP's TCP-like congestion control mechanism, and we suggest that to achieve TCP-fairness, the P2P network's pull-based video segment distribution model should be re-considered carefully.

VI. RELATED WORK

Video streaming is the "killer application" of the P2P network. Many P2P video streaming applications such as GridCast [33], PPLive [19], and UUSee [32] have emerged in the last decade, and some of them become very successful in commerce. In recent years, many efforts have been made to support peer-assisted video streaming on web: providers such as CNTV and Xunlei Kankan [6] rely on their own proprietary protocols and browser plug-ins; while many other providers employ RTMFP [7] developed by Adobe to provide the service. Frontier technologies such as HTML5 WebRTC [34] are also promising for enabling the browser-to-browser video streaming.

For investigating peer-assisted video streaming systems in real world, Hei *et al.* [9] study the performances and user behaviors of PPLive's live video streaming service. Alessandria *et al.* [10] investigate the performances of three commercial IPTV systems under adverse network conditions, and analyze their peer strategies for coping with packet losses and network congestions. Traverso *et al.* [35] experiment and compare the neighbor filtering strategies based on the open-soured WineStreamer P2P-TV system. Zhang *et al.* [6] study the system architecture and performance of the P2P-CDN hybrid video streaming system of Xunlei Kankan. Different from these works that focus on one specific proprietary system, in this paper we focus on RTMFP, which is developed by Adobe as a general-purpose peer-assisted streaming protocol and widely employed by major video service providers, and we select PPTV as an example to study the RTMFP-based video streaming technology. In particular, by reverse-engineering Flash Player, we are capable to perform packet-level studies on RTMFP, which is generally infeasible on proprietary P2P streaming protocols. To the best of our knowledge, this is the first measurement study on RTMFP and RTMFP-based video streaming systems in real world.

It is well recognized that P2P traffics are unfriendly to the traditional TCP-based applications such as FTP and HTTP. To solve this problem, a number of congestion control mechanisms for P2P applications have been proposed in recent years [36][37][25]. In this work, we study the RTMFP protocol that implements its congestion control mechanism in a TCP-like way, and show that to achieve TCP-fairness, a TCP-like congestion control mechanism in P2P protocol is not enough, the inherited video segment distribution model of P2P applications should also be re-considered. As far as we know, this work is the first one to study the TCP-fairness of a peer-assisted video streaming protocol in real world.

VII. Conclusion

In this paper, we investigate Adobe's RTMFP protocol and the RTMFP-based video streaming system of PPTV with a measurement study. We present a methodology for measuring the encrypted RTMFP network and study several key issues of PPTV's RTMFP networks for live and on-demand video streaming. We show that comparing with proprietary P2P networks, the RTMFP network has different system architecture, content distribution policy, potential system bottleneck, and exhibits different degrees of network dynamics.

For data transmission of the RTMFP protocol, we confirm the effectiveness of the protocol's selective retransmission scheme for improving the video playback quality under lossy network conditions. However, there exists unfairness between TCP and RTMFP video streaming traffics, despite that RTMFP implements a TCP-like congestion control mechanism. The unfairness can be ascribed to the mismatch between the P2P network's inherited pull-based segment distribution model and RTMFP's built-in congestion control mechanism; and we suggest that to achieve TCP-fairness, the P2P network's segment distribution model should be re-considered.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 61672486, in part by the sub task of the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDA06011201, and in part by the Anhui Provincial Natural Science Foundation under Grant 1608085MF126.

References

[1] N. Kaushal, "Online videos contribute to 50% of web traffic with Netflix & YouTube being winners!" Nov. 2013, http://www.pagetrafficbuzz.com/online-videos-contribute-50-web-traffic-netflix-youtube-winners/.

[2] S. Gutelle, "Report: 84% of web traffic will be video by 2018," Jun. 2014, http://www.tubefilter.com/2014/06/10/cisco-internet-traffic-rise-2018/.

[3] P. Gill, M. Arlittz, Z. Li, and A. Mahanti, "YouTube traffic characterization: a view from the edge," in *Proc. of IMC'07*, San Diego, CA, USA, Oct. 2007.

[4] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling Netflix: Understanding and improving multi-CDN movie delivery," in *Proc. of IEEE INFOCOM'12*, Orlando, FL, USA, Mar. 2012.

[5] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Communication Survey & Tutorials*, vol. 17, no. 1, pp. 469 – 492, 2015.

[6] G. Zhang, W. Liu, X. Hei, and W. Cheng, "Unreeling Xunlei Kankan: Understanding hybrid CDN-P2P video-on-demand streaming," *IEEE Trans. on Multimedia*, vol. 17, no. 2, pp. 229 – 242, 2015.

[7] M. Thornburgh, "Adobe's secure real-time media flow protocol," IETF RFC 7016, Nov. 2013, https://tools.ietf.org/html/rfc7016.

[8] K. Towes, "P2P on the Adobe Flash platform with RTMFP," Streaming Media East Conference, 2010.

[9] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. on Multimedia*, vol. 9, no. 8, pp. 1672 – 1687, 2007.

[10] E. Alessandria, M. Gallo, E. Leonardi, M. Mellia, and M. Meo, "P2P-TV systems under adverse network conditions: a measurement study," in *Proc. of IEEE INFOCOM'09*, Rio de Janeiro, Brazil, Apr. 2009.

[11] Adobe, "Real-time messaging protocol (RTMP) specification," https://www.adobe.com/devnet/rtmp.html.

[12] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," IETF RFC 5681, Sep. 2009, https://tools.ietf.org/html/rfc5681.

[13] D. Hassoun and J. Heider, "Peer-assisted networking using RTMFP groups in Flash Player 10.1," Feb. 2010, http://www.adobe.com/devnet/adobe-media-server/articles/p2p_rtmfp_groups.html.

[14] "ActionScript 3.0 reference for the Adobe Flash platform," Dec. 2015, http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/.

[15] "The OpenRTMFP project," https://github.com/OpenRTMFP.

[16] C. Sun, "A hook DLL for the RTMFP protocol of Adobe Flash Player," https://github.com/snnn/flash-rtmfp-hook/.

[17] Microsoft Research Asia, "Network emulator toolkit," https://blog.mrpol.nl/2010/01/14/network-emulator-toolkit/.

[18] "Fraps: Real-time video capture & benchmarking," http://www.fraps.com/.

[19] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proc. of ACM SIGCOMM'08*, Seattle, WA, USA, Aug. 2008.

[20] H. He, Y. Zhao, J. Wu, and Y. Tian, "Cost-aware capacity provisioning for Internet video streaming CDNs," *The Computer Journal*, vol. 58, no. 12, pp. 3255 – 3270, 2015.

[21] "G.992.5," https://en.wikipedia.org/wiki/G.992.5.

[22] D. C. Salyers, A. Striegel, and C. Poellabauer, "Wireless reliability: rethinking 802.11 packet loss," in *Proc. of Intl. Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM'08)*, Newport Beach, CA, USA, Jun. 2008.

[23] L. Li, K. Xu, D. Wang, and et al., "A measurement study on TCP behaviors in HSPA+ networks on high-speed rails," in *Proc. of IEEE INFOCOM'15*, Hong Kong, Apr. 2015.

[24] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer internet video broadcast," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 11 – 24, 2008.

[25] M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "NF-TCP: Network friendly TCP," in *Proc. of IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'10)*, Long Branch, NJ, USA, May 2010.

[26] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale CDNs," Microsoft Research, Tech. Rep., 2008.

[27] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64 – 74, 2008.

[28] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. of IEEE INFOCOM'06*, Bacelona, Spain, Apr. 2006.

[29] D. Damjanovic and M. Welzl, "An extension of the TCP steady-state throughput equation for parallel flows and its application in MulTFRC," *IEEE/ACM Trans. on Networking*, vol. 19, no. 6, pp. 1676 – 1689, 2011.

[30] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan, "Modeling short-lived TCP connections with open multiclass queuing networks," *Computer Networks*, vol. 44, no. 2, pp. 153 – 176, 2004.

[31] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communication Surveys & Tutorials*, vol. 12, no. 3, pp. 304 – 342, 2010.

[32] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee: Large-scale operational on-demand streaming with random network coding," in *Proc. of IEEE INFOCOM'10*, San Diego, CA, USA, Mar. 2010.

[33] B. Cheng, L. Stein, H. Jin, X. Liao, and Z. Zhang, "GridCast: Improving peer sharing for P2P VoD," *ACM Trans. on Multimedia Computing, Communications, and Applications*, vol. 4, no. 4, 2008.

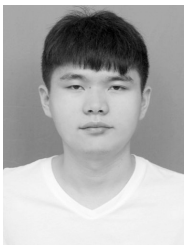
[34] "WebRTC," http://www.webrtc.org/.

[35] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. L. Cigno, and M. Mellia, "Neighborhood filtering strategies for overlay construction in P2P-TV systems: Design and experimental comparison," *IEEE/ACM Trans. on Networking*, vol. 23, no. 3, pp. 741 – 754, 2015.

[36] Y. Liu, H. Wang, Y. Lin, S. Cheng, and G. Simon, "Friendly P2P: Application-level congestion control for peer-to-peer applications," in *Proc. of IEEE GlOBECOM'08*, New Orleans, LO, USA, Nov. 2008.

[37] J. Yan, M. May, and B. Plattner, "Optimal TCP-friendly rate control for P2P streaming: An economic approach," in *Proc. of IFIP Workshop on Self-Organizing Systems (IWSOS'09)*, Zurich, Switzerland, Dec. 2009.

Shan Zou received the bachelor's degree in computer science from the Yunnan University in June 2014. He is a master candidate in the School of Computer Science and Technology, University of Science and Technology of China. His research interest is focused on measuring and evaluating real-world multimedia networking systems. He is a student member of the IEEE.



Qiang Wang received the bachelor's degree in computer science from the Anhui University of Finance and Economics in June 2014. He is currently pursuing the master's degree in the School of Computer Science and Technology, University of Science and Technology of China. His research is focused on the measurement studies of real-world multimedia networks.

Junqiang Ge received the bachelor's degree in computer science from the Central South University in June 2015. He is currently pursuing the master's degree in the School of Computer Science and Technology, University of Science and Technology of China. His research interest is focused on multimedia networks and Internet measurement.

Ye Tian received the bachelor's degree in electronic engineering and the master's degree in computer science from the University of Science and Technology of China (USTC), in July 2001 and 2004, respectively. He received the PhD degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong in December 2007. He is an associate professor in the School of Computer Science and Technology, USTC. He joined USTC in August 2008. His research interests include multimedia networks, Internet measurement, future Internet architectures, and software defined networking. He is a member of the IEEE and ACM, and a senior member of the China Computer Federation. He is currently serving as a young associated editor for *Springer Frontiers of Computer Science Journal*.