IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received September 13, 2019, accepted October 19, 2019, date of publication October 31, 2019, date of current version November 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2950800

Mobile Peer-to-Peer Assisted Coded Streaming

PATRIK J. BRAUN[1,3], ÁDÁM BUDAI[1], JÁNOS LEVENDOVSZKY[2], MÁRTON SIPOS[1], PÉTER EKLER[1], AND FRANK H. P. FITZEK[3,4]

[1]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary
[2]Department of Networked Systems and Services, Budapest University of Technology and Economics, 1111 Budapest, Hungary
[3]Deutsche Telekom Chair of Communication Networks, Technische Universität Dresden, 01069 Dresden, Germany
[4]Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01069 Dresden, Germany

Corresponding author: Patrik J. Braun (patrik.braun@aut.bme.hu)

This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) as a part of Germany's Excellence Strategy - EXC 2050/1 through Cluster of Excellence—Centre for Tactile Internet with Human-in-the-Loop (CeTI) of Technische Universität Dresden under Project 390696704, in part by the National Research, Development and Innovation Fund of Hungary through the FIEK_16 funding scheme, under Project FIEK_16-1-2016-0007, in part by the BME-Artificial Intelligence FIKP of EMMI under Grant BME FIKP-MI/SC, and in part by the János Bolyai Research Fellowship of the Hungarian Academy of Sciences.

ABSTRACT Current video streaming services use a conventional, client-server network topology that puts a heavy load on content servers. Previous work has shown that Peer-to-Peer (P2P) assisted streaming solutions can potentially reduce this load. However, implementing P2P-assisted streaming poses several challenges in modern networks. Users tend to stream videos on the go, using their mobile devices. This mobility makes the network difficult to orchestrate. Furthermore, peers have to contribute their storage to the network, which is challenging, since mobile devices have limited resources compared to desktop machines. In this paper, we introduce an analytical framework for mobile P2P-assisted streaming to estimate the server load that we define as the minimum required server upload rate. Using our framework, we evaluate four caching strategies: infinite cache as a baseline, first in first out (FIFO), random, and Random Linear Network Coded (RLNC) cache. We verify our analytical results with empirical data that was obtained by carrying out extensive measurements on our working P2P system. Our results show that when employing FIFO, random, and RLNC caching strategies, the server load converges to that of the infinite cache as the cache size increases. With a limit of 5 P2P connections per peer, we show that using the random caching, peers can store 40% fewer packets and still achieve the same benefit as with FIFO caching. When using the RLNC caching, it is enough to store 50% fewer packets to achieve the same benefit.

INDEX TERMS Caching, analysis, network coding, peer-to-peer, system implementation, video streaming.

I. INTRODUCTION

Peer-to-Peer (P2P) assisted streaming systems are comprised of a single server and multiple peers. The server is always available, possesses all source packets, and shares them with the peers. A peer downloads packets from the server and shares them with other peers. In this paper, we focus on mobile environments, where peers have limited storage capacity, and the network setup is continuously changing. Therefore, peers have minimum information about the network (i.e., they only know the participants in the network and not what packets they have).

Mobile P2P-assisted streaming has a high potential as video streaming accounted for 60% of the mobile Internet traffic in 2018 [1]. Content providers use extensive server parks, content delivery networks, and other smart caching techniques at Internet Service Providers (ISPs) to serve this huge amount of data [2]. Large Video on Demand (VoD) vendors like BBC iPlayer [3] and Conviva [4] report that peer-assistance has the potential to reduce this server traffic to just 12%.

In addition to the advantages of mobile P2P-assisted streaming, it also poses two significant challenges. 1) Caching on a mobile device is difficult, as it has limited storage capacity compared to a desktop computer. 2) The mobility of the peers makes the network highly dynamic as users watch videos while traveling. Therefore, the set of nearby available peers is continuously changing, which makes connection planning challenging and centralized connection orchestration unfeasible.

These challenges, characteristic of the mobile environment, make it difficult for the content providers to

The associate editor coordinating the review of this manuscript and approving it for publication was Mingjun Dai.

FIGURE 1. **WebPeer protocol running on more than 100 tablets.**

approximate the required server upload rate of a given P2P-assisted service. In order for content providers to adopt P2P-assisted streaming solutions, a mathematical model is needed to predict the behavior (mainly the required server upload rate) of the designed system.

In this paper, we build on our previous work [5] and further extend it by introducing an analytical framework for mobile P2P-assisted streaming. Our proposed framework aims to estimate the server load that we define as the minimum required server upload rate. We consider four different caching strategies, including infinite caching as a baseline, FIFO caching, random caching, and Random Linear Network Coding (RLNC) encoded caching. Furthermore, we investigate various input parameters such as network size, cache size, and whether network coding is applied for our analysis.

We also introduce two protocols for mobile P2P-assisted streaming and Peer-to-Peer-Assisted Streaming Network (*PasNet*), a system that implements those protocols [5]–[7]. We have demonstrated *PasNet*'s practical potential by running it on more than 100 tablets[1] as shown in FIGURE 1. *PasNet* was also presented at several trade shows, including CES'17. We validate the accuracy of our model by comparing its mean square error (MSE) to measurement results that we obtained by running extensive measurements on *PasNet*.

The significance of our contributions with respect to previous works can be summarized in four main aspects: 1) We propose an RLNC coded caching and incorporate it into a P2P protocol. 2) We propose an analytical framework to estimate the server load for mobile P2P-assisted streaming. To the best of our knowledge, this is the first model that incorporates RLNC coded caching and approximates the server load in a mobile P2P-assisted streaming scenario. 3) We validate our framework with real-life measurement results. Most papers in this field present either analytical or practical results, but typically not both. 4) Our results show that random caching outperforms FIFO caching in terms of server load, while RLNC encoded caching reaches the theoretical optimal (achievable with infinite caching), while only caching 20% of the original content.

[1]Video about running *PasNet* on more than 100 tablets: https://www.youtube.com/watch?v=LuGJwkqUyFI

The structure of this paper can be summarized as follows:

- Section II summarizes the related work in the field.
- Section III presents the problem definition of the paper.
- Section IV proposes an analytical framework for estimating the server load.
- Section V presents our four caching strategies, including uncoded and RLNC encoded strategies.
- Section VI introduces two protocols for mobile P2P-assisted streaming: WebPeer protocol uses random caching, while CodedWebPeer employs RLNC encoded caching. Section VI also introduces *PasNet*, our streaming system built on these protocols. To obtain repeatable measurements for this paper, we set up a testbed for *PasNet* using Docker containers and a controlled network environment.
- Section VII evaluates our analytical framework by comparing its results with measurement results from our testbed.
- Section VIII summarizes our findings and possible further research in the field.

II. RELATED WORK

P2P-based content distribution is a widely researched field. Proprietary P2P-based software data distribution is also employed in current systems: as part of Windows Update, in delivering Linux distributions through *BitTorrent* as well as by several other companies such as Peer5 [8]. Work by Chen *et al.* [9] constitutes a significant contribution to understanding user behavior in video streaming applications. They examined *PPLive*, one of the most popular VoD systems in China. Measurements done on a statistically significant group of users suggest that the genre of the video is a key factor when determining how far a user watches the content. Chen *et al.* proposed modeling the time after which users leave the system using a skew-normal distribution and basing its parameter on the genre of the video. We have followed this proposal and incorporated it into our model.

The main difference between a conventional and P2P-assisted streaming service is that peers in a P2P-assisted service need to cache data. For example, in the *PPLive* system, each peer needs to dedicate approximately 1GB of storage [10]. In a resource-limited environment, like smartphones or browser-based applications, it is not always possible to reserve this amount of storage for video streaming. Therefore, a smart caching mechanism is required. Wu and Lui presented mathematical models and formulated an optimization framework to understand the impact of movies' popularities on servers' workload [11]. They proposed a passive and active video replication strategy, where data is passively deleted when the peers' storage is full, while popular content is actively pushed into the storage. They showed that the algorithm is effective even in dynamic environments and movies with different playback rates. Shehab *et al.* presented a P2P video delivery system, where they used the free downlink bandwidth at the peers to prefetch recommended videos according to their interests [12]. They showed through empirical results that their solution could reduce the number of requests to the media server while improving the initial playout latency. Huang *et al.* modeled a mesh-based P2P VoD system [13]. They focused on the problem of how peers should serve requested packets. They proposed Playback-Quality-Aware scheduling that prioritizes the request based on the effects on playback quality at the peer's connections. They showed through simulations that their solution improves the playback quality, but it heavily relies on the honesty of the peers in reporting some key information, such as the urgent property of a given video chunk. In our work, we avoid this issue by proposing a solution where peers are not required to report this key information.

Fujita investigated P2P-assisted delivery networks with multiple trees as the underlying topology of the overlay network [14]. He focused on 2-hop content delivery solutions where the video stream is divided into α stripes, and there are n peers in the network. He showed that if the peers have uniform upload capacity, then n/α upload capacity is sufficient on the server-side to deliver the content to all peers. Karamushuk *et al.* showed that P2P-assisted VoD streaming can achieve a reduction of 88% in server traffic [3], based on measurement data collected using the BBC iPlayer. Mavromoustakis *et al.* investigated P2P streaming in a mobile environment [15] with the focus on modeling the node movement with Fractional Brownian Motion (FBM) and a Random Waypoint Mobility (RWM) model. Their peers used a common look-up table to request specific video streams from other peers. They showed through simulations that FBM gives better overall network performance. In this work, we focus on mobile environments, where it is not possible to maintain a peer connection for an extended period as the set of available peers continuously changes. Furthermore, we also incorporate RLNC to the protocol.

It has been previously shown that coding, particularly Random Linear Network Coding (RLNC) improves the cache hit rate [16] and thereby the overall system performance in Content-Centric Networking (CCN). Furthermore, RLNC has already proved its advantages in other P2P environments [17], [18], even in a limited resource environment, on mobile devices [19].

Our work differs from previous works by focusing on mobile P2P-assisted coded streaming, where the set of available peers is continuously changing, and peers only have minimal information about other peers. Thus, preplanned connection management is not possible. Furthermore, we employ RLNC on the peers' cache. We introduce an analytical framework to investigate the server load regarding employed caching mechanism at the peers. We validate our calculation with extensive measurement results.

III. PROBLEM STATEMENT

We focus on mobile P2P-assisted streaming. FIGURE 2 shows an overview of such a system.

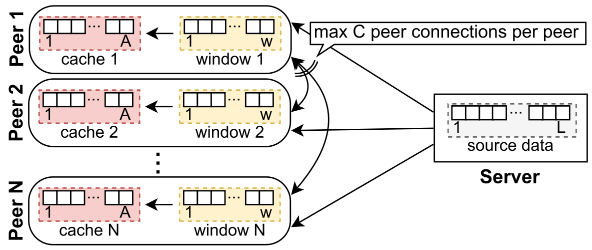

FIGURE 2. System model overview.

We present a discrete-time model. The system contains N_k peers at time k. Peers can continuously join and leave the network, and there is a server that never leaves the network. There is exactly one source file (e.g., a video file), that consists of L number of packets. The server has all L packets. The peers aim to gather all L packets in a streaming way (i.e., downloading them in sequential order).

A. PEER LIFE CYCLE

The lifecycle of a peer can be characterized as follows:

1) *Joining the network:* Peers join the network randomly, following a Poisson process.
2) *Participating in the network:* A newly joined peer does not have any of the L packets. Throughout its life, a peer aims to gather all packets in a streaming way. To do so, it creates connections to other peers and exchanges packets with them. Furthermore, it may also download packets from the server. Peers consume the content (i.e., watch the video) at a constant speed without skipping data (i.e., they do not pause or seek in the video).
3) *Leaving the network:* Peers do not leave the network because of poor quality of service, but they may get bored with the content and leave early. Furthermore, peers may stay in the network after they consumed the content (e.g., the user keeps their browser open after watching a video). Therefore, we also model the peers leaving with a random process. According to Chen *et al.* [9], the duration a peer spends in the network depends on the type of the consumed content and usually follows a skew-normal distribution [20]. We employ this solution to model the peer leaving process.

B. PEER CONNECTIONS

Peers have an up-to-date list of peers in the network. The list only indicates the existence of a peer and no other information is provided (i.e., there is no record about which packets a peer may have). A peer at time k may have connections to a subset of the available $N_k - 1$ peers. The active connections of peer i at time slot k are denoted by vector $\mathbf{c}^{(i)}(k)$ and bounded by C:

$$\mathbf{c}^{(i)}(k) \in \{0,1\}^{N_t}, i = 1, \ldots, N_k \quad w(\mathbf{c}^{(i)}(k)) \le C, \qquad (1)$$

where $w(\mathbf{c}^{(i)}(k))$ gives the active connection count that peer i has to other peers. Peers do not store any historical information about their previous connections, and we assume connection creation and termination to be instantaneous. A peer always has a connection to the server (independent from the C number of peer connections). Connections are created over a perfect channel, so there is no packet loss in the network. Selecting the peers to connect to depends on the employed connection management that we detail in Section IV.

C. DOWNLOAD SCHEDULING

Peers have a download bandwidth π_b (measured in packets) that is equal among all peers. At every time slot k, peers can download up-to π_b number of packets. A peer can download a packet from its connections or the server. Peers aim to minimize the requests to the server: They try to download π_b number of packets from the network. If their P2P connection can only provide $q < \pi_\text{b}$ number of unique packets, they download the rest $\pi_\text{b} - q$ packets from the server.

To force the sequential fashion of the packet download, we assume a w-sized download window (measured in packets) at the peers. Peers are constrained to schedule only those packets for download that are in their download window. If $w = 1$, peers download the packets in sequential order, while if $w > 1$, peers have a chance to download some packets in parallel to improve their throughput. We define $\mathcal{W}^{(i)}(k)$ as the set of packets in the window at peer i at time slot k:

$$\begin{aligned}
\mathcal{L} &= \{1, \dots, L\} \\
\mathcal{L}^{(i)}_{\text{down}}(k) &= \{l \in \mathcal{L} \mid \text{packet } l \text{ was downloaded by time slot } k\} \\
\pi^{(i)}_{\text{p}}(k) &= max(\mathcal{L}^{(i)}_{\text{down}}(k)) \\
\mathcal{W}^{(i)}(k) &= \{l \in \mathcal{L} \mid \pi^{(i)}_{\text{p}}(k) < l \leq \pi^{(i)}_{\text{p}}(k) + w\}, \qquad (2)
\end{aligned}$$

where $\mathcal{L}$ is the set of all packets and $\pi^{(i)}_{\text{p}}(k)$ is highest id of all downloaded packet at peer i at time k. We also refer to $\pi^{(i)}_{\text{p}}(k)$ as the packet-based download progress for peer i at time k.

A peer may download any of the packets in its window in an arbitrary order. The choice depends on the employed download scheduling strategy that we detail in Section IV.

D. PEER CACHE

In a P2P system, peers need to store some data to be able to contribute to the network. However, we focus on mobile environments in this paper, where peers only have limited resources. Therefore, we introduce an A-sized cache (measured in packets) to limit the amount of data stored at the peers. A peer can offer any packet from its cache for download.

We aim to minimize the network overhead at the peers. Therefore, peers only download packets that they need. After a packet download, the peer can choose to cache that packet, based on the employed caching strategy. If the packet was not cached at the time of its download, there is no further chance to cache it later. We call this mechanism *single-try cache*. We detail possible single-try caching strategies in Section V.

TABLE 1. Table of notations.

Symbols	definition
π_t	Peer age
π_b	Available peer bandwidth
$\pi_\text{p}^{(i)}$	Packet-based peer progress of peer i
$\pi_\text{g}^{(i)}$	RLNC generation-based peer progress of peer i
ΔL	Size of a packet
L	Number of packet
N	Number of peers
A	Cache size as ratio
C	Number of P2P connections per peer
w	Size of the download window
g	Generation size, the number of packets on which RLNC is applied
G	Number of RLNC generations
$\Psi(\pi_\text{t})$	Age to progress transformation function
$\overline{r}$	Average server load
U_r	Server upload bandwidth
$s(\pi_\text{p}^{(i)})$	Cache miss ratio with single P2P connection
$\mathbf{S}(\pi_\text{p}^{(i)})$	Cache miss ratio with multiple P2P connections

The possible packets in the cache and in the window at peer i have a distinct set of packets without overlap. We define $\mathcal{A}^{(i)}(k)$ as the set of packets that can be included in the cache (at most A) at peer i at time slot k:

$$\mathcal{A}^{(i)}(k) \subset \mathcal{L}^{(i)}_{\text{down}}(k), \quad w(\mathcal{A}^{(i)}(k)) \leq A, \qquad (3)$$

where $w(\mathcal{A}^{(i)}(k))$ is the number of elements in set $\mathcal{A}^{(i)}(k)$.

E. PROBLEM DEFINITION

This paper aims to minimize the server load that we define as the required minimum server upload rate. We define the server load at time slot k:

$$r(k) = \frac{\text{downloaded packets from the server at time } k}{\text{all downloaded packets at time } k}. \qquad (4)$$

Using $r(k)$, we calculate the average server load $\overline{r}$:

$$\overline{r} = \lim_{T \to \infty} \sum_{t=1}^{T} \frac{1}{T} r(k) \qquad (5)$$

The notation of this paper is summarized in TABLE 1.

IV. ANALYSIS FRAMEWORK

In this section, we present an analytical framework to estimate the average server load $\overline{r}$ of P2P-assisted streaming services. Based on our previous work [5], we distinguish four key parameters that influence the behavior of a P2P-assisted streaming system:

- peer joining and leaving process,
- packet caching strategy,
- connection management,
- download scheduling.

We compose our framework in a modular way so that it can take all four parameters into account. In this paper, we concentrate on different caching strategies, considering the other three parameters as input. As peer-joining and -leaving process, we assume the one that is described in Section III-A.

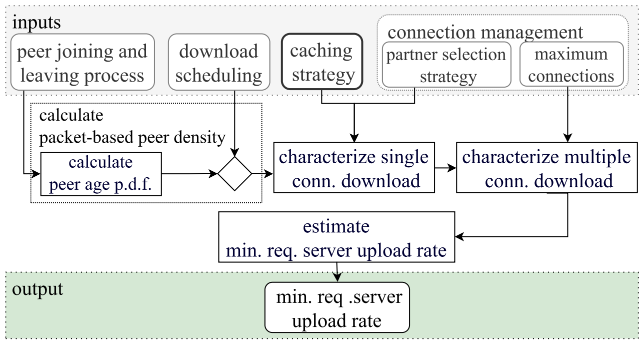
inputs
peer joining and leaving process
download scheduling
caching strategy
connection management
partner selection strategy
maximum connections
calculate packet-based peer density
calculate peer age p.d.f.
characterize single conn. download
characterize multiple conn. download
estimate min. req. server upload rate
output
min. req .server upload rate

FIGURE 3. **Analysis overview.**

Investigating different connection management and download scheduling remains as future work. In this paper, we use empirical results to model them in Section VII. FIGURE 3 gives an overview of our analysis.

In Section III, we presented a discrete-time model. To calculate the average server load $\overline{r}$ for such a system, we would need to evaluate the system at every time slot k. To simplify our calculations, we assume a most likely peer configuration, and we carry out our analysis on this most likely configuration. We define this most likely configuration by using a peerage probability density function (p.d.f.) for a fixed number of peers N. We use our measurement results to validate this assumption in Section VII. Our model has the following four parts:

1) We calculate the peer density as a function of their packet-based progress. With this method, we obtain the ratio of peers (compared to all peers in the network) that have the same packet-based download progress.
2) We calculate the cache miss ratio, assuming the peer has a single P2P connection.
3) We estimate the cache miss ratio if the peer has multiple P2P connections.
4) We use the obtained results to calculate the server load.

A. CALCULATING PACKET-BASED PEER DENSITY

One of the most important aspects of P2P systems is the peers joining and leaving. For streaming, this characteristic mostly depends on the content (i.e., how popular or interesting is the content). Using the Poisson-based peer joining and a skew normal-based leaving process, we calculate the ratio of peers with the same packet-based download progress in relation to all peers (i.e., the number of peers out of N peers that have the same packet-based download progress). We call the vector that contains this ratio for all possible progress as *packet-based peer density*.

To obtain the packet-based peer density ϱ_l for packet l, first we calculate the peer age p.d.f. $f_{\text{alive}}(t)$ that gives the peer density based on peer age (the time a peer spends in the network). Second, we use a transform function $\Psi(t)$ to transform from peerage to peer progress to obtain the peer density based on a download progress scale. Finally, we discretize this into packets-based download progress by partitioning the progress into intervals (e.g., if a peer has progress between 0.1 and 0.2, its packet-based download progress is 2).

1) CALCULATING PEER AGE P.D.F.

Let λ be the intensity of the Poisson process that describes peer joining behavior. Furthermore, let peers leave after a randomly chosen time generated by a skew-normal distribution. The proportion of peers in an arbitrary time interval (Δt) can be calculated in the following way: If the inter-arrival time is short between peers (the value of λ is high), an interval can be constructed in such a way that the survival probability of peers in this interval is approximately equal and the number of peers is significantly larger than 1. We can calculate the mean number of peers in the Δt interval as $n_0\Delta t$, where $n_0 = \lambda$ is the mean density of peers for the Poisson joining process. The probability that a peer survives for at least t time equals $1 - F_{\text{skew}}(t)$, where $F_{\text{skew}}(t)$ is the c.d.f. of the skew normal distribution. In the Δt interval, the number of peers at t follows a binomial distribution. Therefore, the expected number of peers in this small interval is $n_0\Delta t(1 - F_{\text{skew}}(t))$. The peer age p.d.f. $f_{\text{alive}}(t)$ represents the ratio of peers in the small Δt interval compared to the whole system and can be expressed as follows:

$$\begin{aligned} f_{\text{alive}}(t)\Delta t &= \frac{n_0\Delta t(1 - F_{\text{skew}}(t))}{\int_0^\infty n_0(1 - F_{\text{skew}}(t))\mathrm{d}t} \\ &= \frac{\Delta t(1 - F_{\text{skew}}(t))}{\int_0^\infty (1 - F_{\text{skew}}(t))\mathrm{d}t}, \quad t \geq 0. \end{aligned} \tag{6}$$

2) CREATING THE TRANSFORM FUNCTION

A transformation function $\Psi : \pi_{\mathrm{t}} \to \pi_\delta$ can be constructed, where $\pi_{\mathrm{t}}, \pi_\delta \in [0, 1]$ and π_{t} is the duration elapsed since the peer joined the network and π_δ is the download progress of the peer. We consider this function as an input to our model, since the steepness of Ψ represents download speed and its characteristic reflects the properties of the implemented download scheduler (e.g., if peers maintain a constant download speed, its steepness is constant). Ψ has two constraints: (i) it should be strictly monotonically increasing until peers finish downloading and (ii) $0 \leq \frac{\mathrm{d}\Psi}{\mathrm{d}\pi_{\mathrm{t}}} L\Delta L \leq \pi_{\mathrm{b}}$. The former means that a peer can only increase its progress, not decrease it, while the latter limits the download speed not to exceed the available bandwidth at peers. Furthermore, in our analysis, we assume that peers download faster than consuming a given content. Therefore $\Psi(\pi_{\mathrm{t}}) \geq \pi_{\mathrm{t}}$. If we assume that there is no content buffering on the peers side, we can estimate Ψ the following way:

$$\Psi(\pi_{\mathrm{t}}) = \pi_{\mathrm{t}}, \tag{7}$$

otherwise:

$$\Psi(\pi_{\mathrm{t}}) = \min(\pi_{\mathrm{t}} + B, 1), \tag{8}$$

where B is size of the buffer.

3) CALCULATING PACKET-BASED PEER DENSITY

We can obtain progress-based density, $f_{\text{prg}}(n)$ by using the inverse Ψ function:

$$f_{\text{prg}}(n) = f_{\text{alive}}(\Psi^{-1}(n)). \tag{9}$$

Packet-based peer density is given by the following equation:

$$\varrho_l = \int_{\frac{l}{L+1}}^{\frac{l+1}{L+1}} f_{\text{prg}}(n)\mathrm{d}t, \quad l = 0, \ldots, L-1. \tag{10}$$

The packet-based peer density for the peers who have finished downloading is:

$$\varrho_L = \int_{n_{\text{ready}}}^{\infty} f_{\text{prg}}(n)\mathrm{d}n, \tag{11}$$

where $n_{\text{ready}} = \min\{\Psi^{-1}(n) = 1.0\}$ is the time when a peer completed downloading the movie.

B. CHARACTERIZING SINGLE CONNECTION DOWNLOAD

To simplify our analysis, we further assume that the individual connections of a peer are independent. We wish to calculate the cache miss ratio of peer i with $\pi_{\text{p}}^{(i)}$ if it only has a single P2P connection. Two factors influence this: 1) Peers have limited cache size A and drop packets according to their caching strategy. 2) Packets are downloaded in an ordered, streaming manner, meaning that a peer can only download a packet from its download window of size w. We use the notation $\mathbb{E}(\delta_{nm})$ for the expected number of packets obtained by peer i with $\pi_{\text{p}}^{(i)} = n$ from a peer j with $\pi_{\text{p}}^{(j)} = m$, assuming that they are connected. In Section V, we express $\mathbb{E}(\delta_{nm})$ with regards to different caching strategies.

We express the cache miss ratio for peer i with $\pi_{\text{p}}^{(i)} = n$ as follows:

$$s(n) = \sum_{m=0, m\neq n}^{L} \Big(1 - \frac{\mathbb{E}(\delta_{nm})}{\min\{w, L-n\}}\Big)\varrho_n w_p(n, m), \tag{12}$$

where $w_p(n, m)$ is the probability that a peer i with $\pi_{\text{p}}^{(i)} = n$ has an active connection to a peer with packet-based download progress m. Connection management has a significant influence on $w_p(n, m)$, the investigation of different approaches is not part of this paper and remains as future work. During the numerical evaluation of our analysis in Section VII, we estimate $w_p(n, m)$ with measurement-based results.

C. CHARACTERIZING MULTIPLE CONNECTION DOWNLOAD

Since we consider the establishment of connections as independent events, the cache miss ratio for peer i with $\pi_{\text{p}}^{(i)} = n$ can be calculated as follows:

$$\mathbf{S}(n) = \sum_{j=0}^{N}(1 - (1 - s(n))^j)w_c(j), \tag{13}$$

where $w_c(j)$ is the probability that a peer has j connections. Like in the case of $w_p(n, m)$, $w_c(j)$ is also influenced by the used connection management method. Therefore we also estimate $w_c(j)$ with measurement-based results during the numerical evaluation of our analysis.

D. ESTIMATING SERVER LOAD

The average server load that the server must provide to a peer can be estimated in the following way:

$$\overline{r} = \frac{1}{L}\sum_{n=0}^{L}(\mathbf{S}(n))\varrho_n. \tag{14}$$

$\overline{r}$ gives the ratio between the necessary server upload rate and the average download rate per peer. The following formula can be used to obtain the minimum required server upload bandwidth:

$$U_r = \overline{r} N_{\text{d}} \pi_{\text{b}}, \tag{15}$$

where N_{d} is the number of peers that are actively downloading:

$$N_{\text{d}} = N\sum_{n=0}^{L-1}\varrho_n. \tag{16}$$

E. QUALITY OF EXPERIENCE (QOE) ESTIMATION

Our framework also has the potential to be extended with the purpose of investigating the Quality of Experience (QoE) at the peers. It can estimate the probability that a peer does not receive a requested packet in time, leading to interruptions in the video stream and thus poor QoE. This is achieved by calculating the probability that the server cannot respond to a request due to being overloaded. Since peers only download from the server if they cannot collect a packet from the network, any time the server is overloaded, the QoE is affected. Let us assume that the server can serve $\mathcal{C}_{\text{s}}$ number of packets ($\mathcal{C}_{\text{s}}$ peers parallel with one packet request). Furthermore, let $\mathbf{y} \in \{0, 1\}^N$ binary vector represent a given download configuration:

$$y_i = \begin{cases} 1 & \text{if peer } i \text{ downloads from the server} \\ 0 & \text{if peer } i \text{ downloads from the network.} \end{cases} \tag{17}$$

Then, the probability of the number of requests exceeding $\mathcal{C}_{\text{s}}$ server capacity is the following:

$$P(w(\mathbf{y}) > \mathcal{C}_{\text{s}}) = \sum_{y: w(\mathbf{y}) > \mathcal{C}_{\text{s}}} \prod_{i=1}^{N} \mathbf{S}(\pi_{\text{p}}^{(i)})^{y_i}(1 - \mathbf{S}(\pi_{\text{p}}^{(i)}))^{1-y_i}, \tag{18}$$

where $w(\mathbf{y})$ is the number of 1-s in vector $\mathbf{y}$.

This method is a good indicator of the expected QoE. However, it does not incorporate all aspects that lead to possible stream interruptions. A more thorough model should investigate how peers buffer the video and how the server chooses which request to the server when overloaded. Therefore, a detailed characterization of the QoE in a mobile P2P-assisted system is left as future work.

V. SINGLE-TRY CACHING STRATEGIES

In a *single-try cache*, after a packet is downloaded, a peer tries to cache that packet. If the packet was not cached at the time of its download, there is no further chance to cache it later. Therefore, conventional caching, like Least Recently Used (LRU) [21], cannot be applied to our model.

In this section, we elaborate on four single-try caching strategies: infinite caching, FIFO caching, random caching, and RLNC encoded caching.

A. INFINITE CACHING

We propose an infinite caching approach to serve as a baseline in our analysis. The infinite caching is capable of caching all downloaded packets.

The expected number of packets obtained by peer i with $\pi_{\mathrm{p}}^{(i)} = n$ from a peer j with $\pi_{\mathrm{p}}^{(j)} = m$, $m \geq n$ is:

$$\mathbb{E}(\delta_{nm}) = \min\{w, m-n\} \tag{19}$$

B. FIFO CACHING

A simple caching strategy is to store the last A packets. In this case, the expected number of packets obtained by peer i with $\pi_{\mathrm{p}}^{(i)} = n$ from a peer j with $\pi_{\mathrm{p}}^{(j)} = m$, $m \geq n$ is:

$$\begin{aligned} w_{\text{start}}^{(n)} &= n \\ w_{\text{end}}^{(n)} &= \min\{n+w, L\} \\ A_{\text{start}}^{(m)} &= \max\{0, m-A\} \\ A_{\text{end}}^{(m)} &= m \\ \mathbb{E}(\delta_{nm}) &= \max\{0, \min\{A_{\text{end}}^{(m)} - w_{\text{start}}^{(n)}, w_{\text{end}}^{(n)} - A_{\text{start}}^{(m)}\}\}, \end{aligned} \tag{20}$$

where $A_{\text{start}}^{(m)}$ and $A_{\text{end}}^{(m)}$ represented the start end the end of the cache at peer j and $w_{\text{start}}^{(n)}$ and $w_{\text{end}}^{(n)}$ represents the start and the end of the window of peer i.

Note that this approach may provide a close-to-optimal solution in a more general P2P-assisted streaming scenario, where peer i with $\pi_{\mathrm{p}}^{(i)} = n$ would be able to find a peer j with $\pi_{\mathrm{p}}^{(j)} = m$, $m > n$, $m - n < A$ and stick to that peer throughout their download. In that scenario, peers would be able to create a series of peers where each peers downloads packets from the peer in front of it, while only the first peer would download from the server, as it was shown by Do *et al.* [22].

C. RANDOM CACHING

We also propose a random caching strategy that tries to cache packets uniformly across the downloaded data. We achieve this by using the following algorithm: In this caching strategy, peers keep every downloaded packet with the same probability. If the peer progress is $\pi_{\mathrm{p}}^{(i)} \leq A$, the peer i keeps all packets in its cache. The stored packets on peer i can be represented with the following vector $\mathbf{v} \in \{0,1\}^L$, where each element is the probability that peer i has that particular packet:

$$\begin{aligned} &\pi_{\mathrm{p}}^{(i)} = A: \\ &\mathbf{v}_A = \begin{bmatrix} 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 \end{bmatrix}. \end{aligned} \tag{21}$$

Every time peer i exceeds its available cache size, it has to delete exactly one packet. If the peer progress is $\pi_{\mathrm{p}}^{(i)} = A+1$, the peer i discards one packet randomly with a uniform distribution, creating the following vector:

$$\begin{aligned} &\pi_{\mathrm{p}}^{(i)} = A+1: \\ &\mathbf{v}_{A+1} = \begin{bmatrix} \frac{A}{\pi_{\mathrm{p}}^{(i)}} & \frac{A}{\pi_{\mathrm{p}}^{(i)}} & \dots & \frac{A}{\pi_{\mathrm{p}}^{(i)}} & 0 & 0 & \dots & 0 \end{bmatrix}. \end{aligned} \tag{22}$$

At every step $\pi_{\mathrm{p}}^{(i)} > A+1$, peer i deletes an old packet with probability $\frac{1}{\pi_{\mathrm{p}}^{(i)}}$ and the newly downloaded packet with probability $\frac{\pi_{\mathrm{p}}^{(i)} - A}{\pi_{\mathrm{p}}^{(i)}}$:

$$\begin{aligned} &\text{step } \pi_{\mathrm{p}}^{(i)}, \\ &\mathbf{v}_{\pi_{\mathrm{p}}^{(i)}} = \begin{bmatrix} \frac{A}{\pi_{\mathrm{p}}^{(i)}} & \frac{A}{\pi_{\mathrm{p}}^{(i)}} & \dots & \frac{A}{\pi_{\mathrm{p}}^{(i)}} & 0 & 0 & \dots & 0 \end{bmatrix} \\ &\text{step } \pi_{\mathrm{p}}^{(i)} + 1, \\ &\mathbf{v}_{\pi_{\mathrm{p}}^{(i)}+1} = \left[\frac{A}{\pi_{\mathrm{p}}^{(i)}}\left(1 - \frac{1}{\pi_{\mathrm{p}}^{(i)}}\right) \;\dots\; \frac{A}{\pi_{\mathrm{p}}^{(i)}}\left(1 - \frac{1}{\pi_{\mathrm{p}}^{(i)}}\right) \right. \\ &\qquad\qquad \left. 1\left(1 - \frac{\pi_{\mathrm{p}}^{(i)} - A}{\pi_{\mathrm{p}}^{(i)}}\right) \; 0 \; \dots \; 0 \right] \\ &\qquad = \begin{bmatrix} \frac{A}{\pi_{\mathrm{p}}^{(i)}+1} & \dots & \frac{A}{\pi_{\mathrm{p}}^{(i)}+1} & \frac{A}{\pi_{\mathrm{p}}^{(i)}+1} & 0 & \dots & 0 \end{bmatrix}. \end{aligned} \tag{23}$$

Using this algorithm, peer i will store packet l with $max(\frac{A}{\pi_{\mathrm{p}}^{(i)}}, 1)$ probability, where $l <= \pi_{\mathrm{p}}^{(i)}$.

The probability that peer i with $\pi_{\mathrm{p}}^{(i)} = n$ obtains q number of packets from a peer j with $\pi_{\mathrm{p}}^{(j)} = m$, $\pi_{\mathrm{p}}^{(j)} > \pi_{\mathrm{p}}^{(i)}$ (assuming that they are connected) is given as:

$$P_{nm}(q) = \frac{\binom{\min\{w, m-n\}}{q}\binom{\max\{n, m-w\}}{A-q}}{\binom{m}{A}}. \tag{24}$$

Thus, the expected number of packets obtained by peer i with $\pi_{\mathrm{p}}^{(i)} = n$ from a peer j with $\pi_{\mathrm{p}}^{(j)} = m$ is:

$$\mathbb{E}(\delta_{nm}) = \sum_{q=0}^{\min\{A, w, m-n\}} P_{nm}(q) q. \tag{25}$$

D. RLNC CACHING

As an alternative for packet-level caching and data handling, we apply network coding on the data. First, we group the original L packets into g-sized groups, so-called *generations*. There are altogether $G = \lceil L/g \rceil$ *generations*. Then, we use Random Linear Network Coding (RLNC) on these *generations*: each *generation* goes to an RLNC coder that creates encoded packets, by creating linear combinations of those packets with a randomly chosen coefficient. An RLNC coder has a rank that is a measure of the number of linearly independent packets it contains. Each coder starts empty, with rank 0. RLNC coders also support recoding, i.e., creating new linear combinations from already collected packets, without the need of having a full rank coder.

As peers work on a generation-level instead of a packet-level, the RLNC caching strategy works the following way: to keep our calculation simple, we still assume a uniformly distributed cache, but we reduce its variance with the following modification: packets are deleted in such a way, that each *generation* at peer i contains at least $\lfloor A/\pi_{\mathrm{g}}^{(i)} \rfloor$ packets, where $\pi_{\mathrm{g}}^{(i)}$ is the generation-based progress of peer i. Peer deletes packets x from a generation by creating $g-x$ recoded packets and keeping those recoded packets. This technique ensures that after deletion, each peer has a unique set of encoded packets. Then, the other $A - \lfloor A/\pi_{\mathrm{g}}^{(i)} \rfloor$ packets are selected uniformly at random to keep with the method presented above, in such a way that each *generation* contains at most $\lceil A/\pi_{\mathrm{g}}^{(i)} \rceil$ packets. Intuitively, this strategy helps peers to find packets in the network with higher probability, thus lowering cache miss ratio compared to random caching strategy. Furthermore, since peers store encoded data, they also request packets from a given generation, instead of requesting individual packets. Peer i can request any generation from its window. The generation-based window has altogether $w' = \left\lfloor \frac{w}{g} \right\rfloor$ generations. The following generations are in the window of peer i: $[\pi_{\mathrm{g}}^{(i)} + 1, \pi_{\mathrm{g}}^{(i)} + 1 + w']$, where $\pi_{\mathrm{g}}^{(i)} = \lfloor \pi_{\mathrm{p}}^{(i)}/g \rfloor$ is generation-based progress of peer i.

Because of the modification presented above, our analysis should be updated to work on a generation-based instead of a packet-based scale. We use "′" to mark the updated notations:

First ϱ', the generation-based peer density should be calculated:

$$\varrho'_l = \int_{\frac{l}{G+1}}^{\frac{l+1}{G+1}} f_{\mathrm{prg}}(n)\,\mathrm{d}n,\ l = 0, \dots, G-1$$
$$\varrho'_G = \int_{n_{\mathrm{ready}}}^{\infty} f_{\mathrm{prg}}(n)\,\mathrm{d}n, \tag{26}$$

where $n_{\mathrm{ready}} = \min\{\Psi^{-1}(n) = 1.0\}$ is the time when a peer completed downloading the movie.

Second, all generations in the cache of peer j, $\pi_{\mathrm{g}}^{(j)} = m'$ contain $g_{\min}^{m'} = \min\left\{\left\lfloor \frac{A}{m'} \right\rfloor, g\right\}$ packets, and some of them $g_{\min}^{m'} + 1$ packets. Therefore, peer j can serve at least $g_{\min}^{m'}$ packets per generation to peer i, and $g_{\min}^{m'} + 1$ packets per generation with some probability, assuming $\pi_{\mathrm{g}}^{(j)} > \pi_{\mathrm{g}}^{(i)}$. Using this, we express $P'_{n'm'}$, the probability that a peer j with $\pi_{\mathrm{g}}^{(j)} = m'$ can serve a packet from q number of generations with rank $= g_{\min}^{n'} + 1$ to a peer i with $\pi_{\mathrm{g}}^{(i)} = n'$ in the following way:

$$P'_{n'm'}(q) = \frac{\binom{\min\{w', m'-n'\}}{q}\binom{\max\{n', m'-w'\}}{A_m - q}}{\binom{m'}{A_m}}, \tag{27}$$

where $A_m = A \bmod m'$ and has an expected value of:

$$\mathbb{E}(P'_{n'm'}) = \sum_{q=0}^{\min\{A_m, w', m'-n'\}} P'_{n'm'}(q)q. \tag{28}$$

Third, using the new $\mathbb{E}(P'_{n'm'})$, the expected number of packets that a peer j with $\pi_{\mathrm{g}}^{(j)} = m'$ can provide to peer i with $\pi_{\mathrm{g}}^{(i)} = n'$ can be obtained:

$$\mathbb{E}'(\delta_{n'm'}) = \begin{cases} \min\left\{\frac{A}{n'}, w\right\} & \text{if } g \geq w \\ \min\{w', m'-n'\}g_{\min}^{m'} + & \\ \mathbb{E}(P'_{n'm'}) \text{ otherwise.} & \end{cases} \tag{29}$$

Building on the new $P'_{n'm'}(q)$ and $\mathbb{E}'(\delta_{n'm'})$, we can express $s'(n')$, the proportion of data that has to come from the server in case of a peer with a single connection:

$$s'(n') = \sum_{\substack{m'=0 \\ m' \neq n'}}^{G} \left(1 - \frac{\mathbb{E}'(\delta_{n'm'})}{\min\{w, L - n'g\}}\right) \varrho'_{n'} w'_p(n', m'), \tag{30}$$

where $w'_p(n', m')$ gives the probability of a peer i with $\pi_{\mathrm{g}}^{(i)} = n'$ has a connection to a peer with generation-based download progress m'. Like in the case of $w_p(n, m)$, we estimate the $w'_p(n', m')$ with measurement-based results during the numerical evaluation of our analysis in Section VII.

Since peers store unique recoded packets in their RLNC caching, if two peers can serve x number of packets, then it is highly likely that $2x$ useful packets can be gathered. This is in contrast to the random caching, where the available packets have a high probability of overlapping, so only $\leq 2x$ packets are useful (at this point we assume, that the finite field used for network coding is large enough for the probability of generating linearly dependent packets to be negligible [23]). Using this, $\mathbf{S}'(n')$ can be expressed:

$$\mathbf{S}'(n') = \sum_{m'=0}^{N} (1 - \min\{(1 - s'(n'))m', 1\}) w'_c(m'). \tag{31}$$

where $w'_c(j)$ gives the probability that peer i with $\pi_{\mathrm{g}}^{(i)} = n'$ has j connections. Like in the case of $w_c(n', m')$, we estimate the $w'_c(j)$ with measurement-based results during the numerical evaluation.

Finally, we can apply $\mathbf{S}'(n')$ to obtain U'_r:

$$\bar{r}' = \frac{1}{G}\sum_{n'=0}^{G} (\mathbf{S}'(n')\varrho'_{n'})$$
$$U'_r = r' N'_{\mathrm{d}} \pi_{\mathrm{b}}$$
$$N'_{\mathrm{d}} = N \sum_{n'=0}^{G-1} \varrho'_{n'}, \tag{32}$$

The formula shows that by increasing the generation size g, the expected number of packets that a peer can serve $\mathbb{E}'(\delta_{n'm'})$ also increases, thereby decreasing the required server upload rate U'_r.

One should also note, that if $\left\lfloor \frac{A}{\pi_{\mathrm{g}}^{(i)}} \right\rfloor \geq w$, i.e. each cached generation contains more packets than the size of the download window, then a peer with $\pi_{\mathrm{g}}^{(i)}$ will be able to serve all

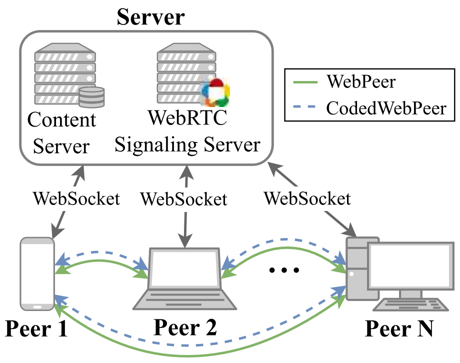

FIGURE 4. System overview.

requested packets for a given download window. Furthermore, if $g = 1$ the U'_r in the case of RLNC encoded caching equals to U_r with random caching. However, increasing g increases the computational overhead and decoding delay, which should be kept low.

VI. TECHNICAL DESCRIPTION OF THE P2P PROTOCOLS AND OUR SYSTEM

We have designed a system, Peer-to-Peer-Assisted Streaming Network (PasNet) to validate our model's accuracy. *PasNet* supports two protocols for P2P-assisted browser-based streaming: 1) *WebPeer* implements the random caching for uncoded data distribution that is presented in Section V-C. 2) *CodedWebPeer* implements RLNC caching for RLNC encoded data distribution that is presented in Section V-D. We chose a web environment for our system to run on multiple platforms, including mobile phones. *PasNet* and both protocols are designed to fulfill the requirements of a mobile P2P-assisted streaming system, as presented in Section III and IV. Building on *PasNet*, we have created a testbed to measure key network characteristics such as server download and upload rate and the streaming progress of peers. FIGURE 4 shows an overview of *PasNet*.

A. THE WEBPEER AND CODEDWEBPEER PROTOCOLS

Both *WebPeer* and *CodedWebPeer* are BitTorrent-like [24] communication protocols, designed specifically for mobile P2P-assisted streaming over Web Real-Time Communication (WebRTC). The *WebPeer* protocol defines the following messages:

1) *Availability vector*: Similarly to BitTorrent's bitfield, it represents the packets in a peer's cache.
2) *Have*: Signals if a new packet is available in the peer's cache.
3) *Lost*: Signals if a packet was deleted from the peer's cache.
4) *Request*: Peer requests a given packet from its connection.
5) *Cancel*: Cancels a request.
6) *Data*: Contains the requested data packet.

All messages contain a packet ID. Once a connection has been made between two peers, they exchange their *availability vector*. Later, they only use *have* and *lost* messages to indicate the changes of their *availability vectors*.

CodedWebPeer uses the same messages as *WebPeer* with some modifications. The protocol handles *generation* indices rather than packet indices. Therefore, each *have* message contains a generation index and the rank of the given coder. The *lost* message is not supported, since sending a *have* message with the decreased rank serves this purpose. *Request* and *cancel* messages contain a value beside the generation index that specifies the number of requested or canceled packets from a given *generation*.

B. OUR SYSTEM: PASNET

PasNet consists of a server and several peers. Peers use *WebRTC* to create P2P connections and either *WebPeer* or *CodedWebPeer* to exchange data. The purpose of the server is twofold: (i) it serves as a content distributor and (ii) acts as a signaling service for *WebRTC*. The signaling service is an intermediary when creating P2P connections and also maintains a list of online peers. To make the system as distributed as possible with minimal central control, the server does not keep track of the amount of data stored on individual peers and does not provide any system–level information, besides listing online peers. The system has been developed in *Typescript* and compiled to *Javascript*. Peers run in the browser, while on the server-side, *Node.js* is used. To perform network coding calculations in an efficient manner, we employed *KODO* [25], an open-source C++ library that supports different finite fields, including GF (2), GF $\left(2^8\right)$ and GF $\left(2^{16}\right)$. We used *Emscripten*[2] to compile the C++ source to a single *JavaScript* file.

1) CONNECTION MANAGEMENT

Peers are limited to C active connections. Each connection is bidirectional, behaving in the same way, regardless of the initiator. Peers accept all incoming connection. Peers aim to keep their active connection count between $C-1$ and C using Algorithm 1.

If a peer has less than C connections, it tries to establish a new one. It chooses a new partner randomly with an equal chance from the list of available peers. If a peer has more than $C - 1$ connections, it closes one of the slowest connection, until it has $C - 1$ connections. New connections take a few seconds to establish and must not be terminated during this time to give them a chance to speed up. These rules ensure that connections are continuously rotated (without thrashing), creating the possibility of finding peers with the highest upload rates. After a peer has finished downloading, it may stay in the network for some time to finish watching the content. During this phase, it does not initiate new connections or close existing ones but accepts new incoming connections.

[2]Emscripten: https://emscripten.org/

Algorithm 1 Connection Management Cycle of Peer i

```
function ConnectionLifeCycle( )
    while true do
        while $activeConnectionCount < C$ do
            CreateNewConnection( )
        end while
        while $activeConnectionCount > C - 1$ do
            RemoveAConnection( )
        end while
    end while
end function
function RemoveAConnection( )
    SortByDownloadSpeedASC(activeConnections)
    for each connection ∈ activeConnections do
        if connection.isNew == False then
            terminate(connection)
            return
        end if
    end for
end function
```

2) DOWNLOAD SCHEDULING

As described in Section III, peers maintain a w-sized window to schedule packets downloads from. The overall aim of the download scheduler is to maximize download speed while keeping the number of packets that are requested from the server to a minimum. It only schedules a packet to be downloaded from the server if none of the connected peers have it. Furthermore, it distributes request across the connected peers based on their offered rate. The number of packets scheduled for download from a peer is linearly proportional to its offered rate. It has been previously shown in [26] that this kind of connection parallelization helps improve throughput while avoiding congestion.

3) RLNC LINEARLY DEPENDENT PACKET DETECTION

In the case of network coding, peers only know the rank of RLNC coders at their connected peers, but not whether they contain any useful packets or not. E.g., peer i and peer j both have 5 out of 10 packets in encoder n', they don't know if those five packets are linearly independent or not until they download them. Our current implementation only supports linearly dependent packet detection, not avoidance. A peer can determine if their partner has useful packets in a given generation only after it has received a packet from that given generation. Peers only send recoded packets, so if a peer receives a linearly dependent packet, it is highly likely that the following packets from the same generation will also be linearly dependent[3]. If peer i receives two linearly dependent packets from generation n' from peer j while rank(n') at j is constant, peer i will *freeze* that generation and will not request any further packets from j. If rank(n') increases at j, peer i will *unfreeze* the generation and try to request further packets.

[3] *RLNC* may also generate linearly dependent packets by unfortunately chosen coefficients. The probability of this happening decreases for larger the finite fields.

C. TESTBED

We have created a testbed for measuring the capabilities of *PasNet* that emulates peer behavior. Peers join the network randomly following a Poisson process and leaves the network following a skew-normal distribution, the parameter of which is influenced mainly by the content type. Our emulated peers also leave the network according to this. Our testbed can measure key network characteristics such as the average online peer count, peer and server download and upload rate, streaming progress, and connection count per peer. It can also be used to evaluate the distribution of these metrics based on the peers' age, among other things. Furthermore, it also tracks the data cached on each peer on a per-packet level.

Our testbed can represent two scenarios:

- peers download from both the server and the P2P network using the *WebPeer* protocol,
- peers download from the server and the P2P network using the *CodedWebPeer* protocol.

Since this paper focuses on approximating the server load in a *PasNet*-like system, our testbed can measure this parameter as well. We collect two types of data for this purpose: First, peers track cache hit ratio: the ratio of packets that peers can potentially download from the P2P network (i.e., their connections posses those packets). We use this to calculate the cache miss ratio (i.e., the ratio of packets that peer's connections do not possess). Based on this, we calculate a metric that we refer to as the *achievable server load*. Second, peers also gather information about the ratio of packets downloaded from the server. We use that to calculate another metric that we refer to as the *effective server load*. The *effective server load* is always greater or equal to the *achievable server load*. We use the *achievable server load* to validate our model. The difference between the two metrics may stem from several sources; we have tried to identify the most important ones. Errors are unavoidable in the estimations that peers make based on historical data about the available bandwidth of their connections. This estimation is used to schedule packets for download and may lead to suboptimal results. If a peer leaves, it takes time to propagate this information in the network depending on the keep-alive timeout of WebRTC. This delay may also cause scheduling issues as peers request unavailable packets. Kernel-level packet queuing or at the bandwidth shaper of the testbed can cause packet delay, leading to a race condition. Furthermore, peers use the same channel for data transfer and the exchange of control messages (like *have*, *cancel*,etc.). Thus, data transfer may use up significant bandwidth and cause peers to have outdated information about their connections.

We run our testbed on a headless Ubuntu server using Docker[4] and FireQOS[5] to create a realistic network setup

[4] Docker url: https://www.docker.com/

[5] FireQOS url: https://firehol.org/

with real bandwidth constraints and packet drops. Each peer and the server runs in a separate Docker container. Google Chrome is used to run the peers in headless mode.

VII. EVALUATION RESULTS

In this section, we present a numerical evaluation for our analysis and compare it to measurements from our testbed. As presented in Section IV, our formulas use connection management and download scheduling as an input parameter. To have a fair comparison between the analytical and the empirical results and to better reflect a real-life P2P system, we derive those input from our measurement results: $w_p(n, m)$, the probability that peer i with $\pi_{\mathrm{p}}^{(i)} = n$ has an active connection to peer j with $\pi_{\mathrm{p}}^{(j)} = m$ is specified as follows:

$$w_p(n,m) = \begin{cases} +0.00003\frac{n}{L} + 0.0572 \text{ if } 0.0 \le m \le 0.1 \\ -0.00005\frac{n}{L} + 0.0875 \text{ if } 0.1 < m \le 0.2 \\ -0.00013\frac{n}{L} + 0.1213 \text{ if } 0.2 < m \le 0.3 \\ -0.00012\frac{n}{L} + 0.1213 \text{ if } 0.3 < m \le 0.4 \\ -0.00014\frac{n}{L} + 0.1376 \text{ if } 0.4 < m \le 0.5 \\ -0.00007\frac{n}{L} + 0.0949 \text{ if } 0.5 < m \le 0.6 \\ -0.00009\frac{n}{L} + 0.1144 \text{ if } 0.6 < m \le 0.7 \\ -0.00009\frac{n}{L} + 0.1017 \text{ if } 0.7 < m \le 0.8 \\ -0.00013\frac{n}{L} + 0.1546 \text{ if } 0.8 < m \le 0.9 \\ +0.00005\frac{n}{L} + 0.1147 \text{ if } 0.9 < m \le 1.0. \end{cases} \quad (33)$$

$w_c(j)$, the probability that a peer has j connections is specified as follows:

$$w_c(j) = \begin{cases} 0.02 & \text{if } j = 3 \\ 0.16 & \text{if } j = 4 \\ 0.48 & \text{if } j = 5 \\ 0.25 & \text{if } j = 6 \\ 0.07 & \text{if } j = 7 \\ 0.02 & \text{if } j = 8 \\ 0, & \text{otherwise.} \end{cases} \quad (34)$$

Furthermore, FIGURE 5 presents a measured transformation function, $\Psi(t)$ (showing its 5,25,50,95 percentile). 50 percentile of the peers finished downloading the content after 120s and some stayed in the network to contribute for an additional 250s. In our calculations, we use the 50 percentile of the measured function as $\Psi(t)$. We obtain the numerical results in this section by using $L = 633$ packets, $N = 18$ peers that could create up to $C = 5$ P2P connections.

The rest of this section follows the structure of Section IV, and we evaluate each step separately.

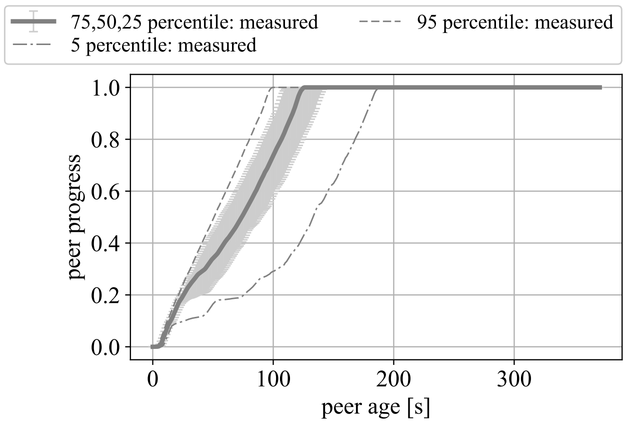

FIGURE 5. Empirical transformation function, $\Psi(t)$.

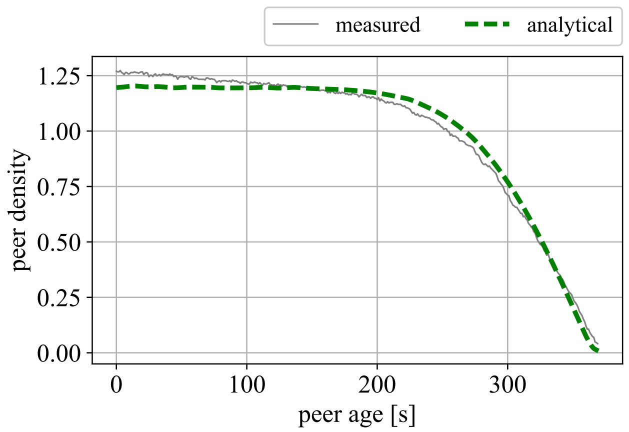

FIGURE 6. Peer count probability density function $f_{\text{alive}}(t)$.

A. PACKET-BASED PEER DENSITY

We used a Poisson process to model peer arrival Peers leave after a randomly chosen time generated by a skew-normal distribution, as described in Section IV. To obtain the packet-based peer density, first we calculate peer age p.d.f. that is presented in FIGURE 6. Our computed empirical peer age p.d.f. distribution shows similar trends with 0.149 mean square error (MSE). The figure shows that most of the peers stayed in the network for 250s, which is about twice the duration of downloading the content. After 250s, peers started leaving the network, and the last one left after about 370s.

Using the calculated peer age p.d.f. and the transformation function $\Psi(t)$ from FIGURE 5, we get the packet-based peer density that we use in our subsequent calculations. Note that, because of the shape of the peerage p.d.f., there are more young peers than old peers. This behavior leads to first packets having a higher probability to be found in the network than the last packets.

B. SINGLE-CONNECTION DOWNLOAD

In FIGURE 7, we present results for downloading with a single peer connection with cache size $A = 0.125\,L$. We compare the modeled infinite, FIFO, and random caching strategies to WebPeer (random caching implementation)-based

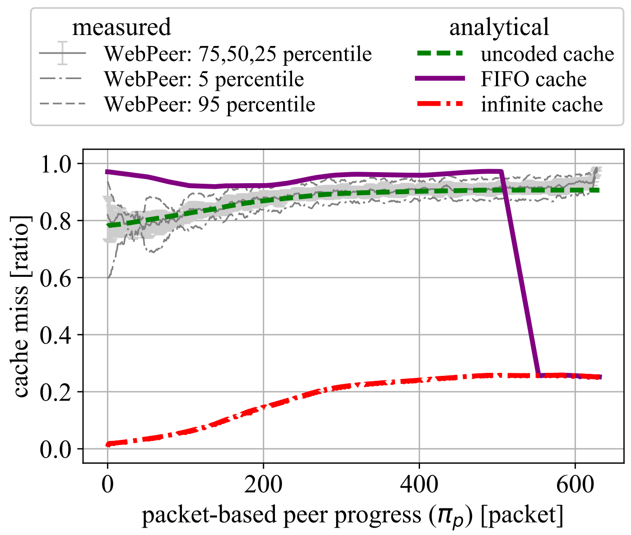

FIGURE 7. Comparing measured and analytical results of cache miss ratio with a single P2P connection $s(n)$. Cache size $A = 0.135L$, peers $N = 18$ and packets $L = 631$.

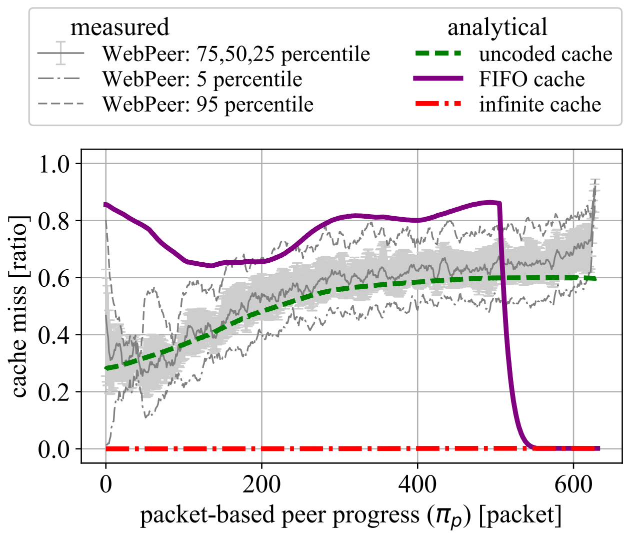

FIGURE 8. Comparing measured and analytical results of cache miss ratio with multiple P2P connections $\mathbf{S}(n)$. Cache size $A = 0.125L$, peers $N = 18$ and packets $L = 633$.

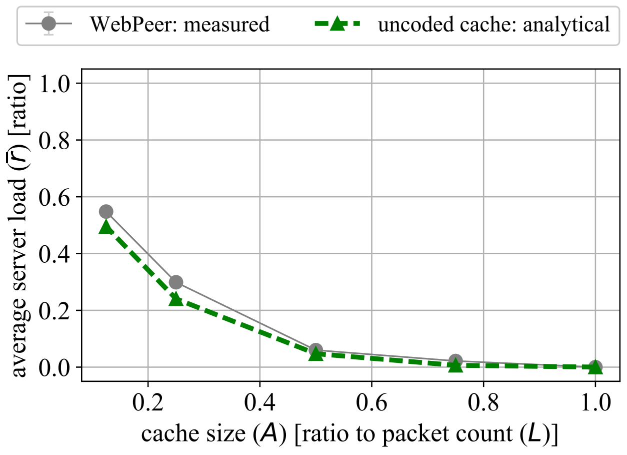

FIGURE 9. Achievable server load r compared to empirical *WebPeer* achievable server rate with peers $N = 18$ and packets $L = 633$.

measurement results. The line depicting the estimation of the random caching analysis is within the 25 and 75 percentile and close to the 50 percentile of the WebPeer, and it has MSE of 0.01216. The figure shows that the cache miss ratio with random caching is about 0.78 for the first few packets. This ratio reaches 0.9 when a peer aims to download the second half of the packets. The slight P2P performance increase at the beginning of the download stems from the fact that more peers have downloaded the beginning than the end of the file. We also compared our random caching analysis to WebPeer measurement results of cache sizes $A \in \{1, 0.75, 0.5, 0.25\}L$. Our solution provides a good approximation for all cases with $\leq$ 0.045 MSE. FIGURE 7 also shows that FIFO caching performs poorly, when a peer downloads packets with id $< L - A$. The reason is that there are a significant amount of peers that have already finished downloading the content and are waiting in the network. These peers keep the last A packets in their cache. Once a peer starts downloading packets with id $\geq L - A$, FIFO caching performs as the infinite caching that gives the best theoretical performance.

C. MULTI-CONNECTION DOWNLOAD

FIGURE 8 presents the ratio $\mathbf{S}(n)$ of packets of peer i with $\pi_p^{(i)}$ needs to download from the server if it has multiple P2P connections.

Random caching shows similar trends to the measured WebPeer. We also compared them with different cache sizes $A \in \{1, 0.75, 0.5, 0.25\}L$. They show high correlation with $\leq$ 0.089 MSE. As seen in the figure, $\mathbf{S}(n)$ for the random caching has a similar trend to $s(n)$. Using multiple connections amplifies the effect of the first packets being more likely to be found in the network than the last packets. Therefore the young peers who download the beginning of the content are likely to use less server resources.

Using FIFO caching strategy with multiple P2P connections also improves performance, but the random caching strategy has a more significant increase. FIFO caching reaches the performance of infinite caching with a single connection only when a peer downloads packets with id $< L - A$. Using infinite caching with multiple connections, P2P connections can serve all requested packets. Therefore, peers do need to download any packets from the server.

D. SERVER UPLOAD RATE WITH RANDOM CACHE

FIGURE 9 compares our empirical r from WebPeer with our random caching analysis. They show similar trends. As the cache size shrinks, the error of our analysis increases slightly, never exceeding 0.05 (out of a maximum of 1.0), corresponding to an error between the analysis of r and measured WebPeer's r of less than 5%. Furthermore, results show that if peers cache only $0.125L$ packets, the server needs to contribute half of the data. In contrast to that,

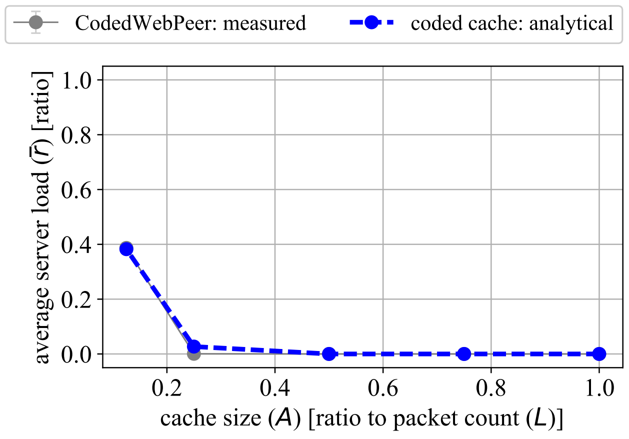

FIGURE 10. Achievable server load r' compared to empirical *CodedWebPeer* achievable server load with peers $N = 18$, generation size $g = 8$ and packets $L = 633$.

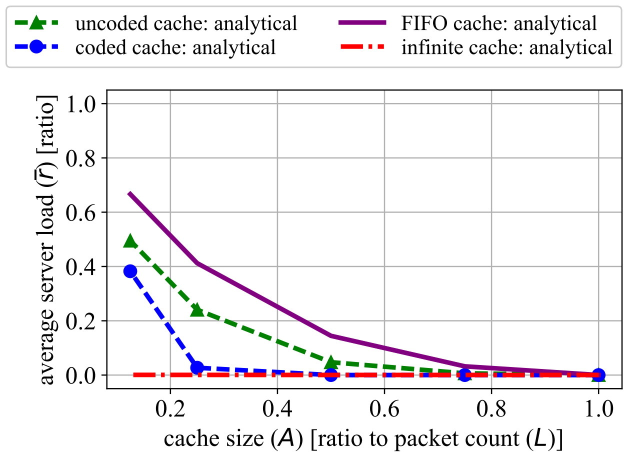

FIGURE 11. Comparing server load for all four caching strategies with peers $N = 18$, generation size $g = 8$ and packets $L = 633$.

having $A \geq 0.75L$, the server load approaches 0. This is much earlier than random or FIFO caching.

E. APPLYING RANDOM LINEAR NETWORK CODING

We also calculated the server load r' with RLNC caching. FIGURE 10 compares our empirical CodedWebPeer results r' with our RLNC caching analysis. The figure shows that applying network coding improves network performance significantly as peers need to download fewer packets from the server. Using cache size $A = 0.125L$, only 40% of the data comes from the server, in contrast to 50% without network coding. Furthermore, having $A \geq 0.3L$, the server load already approaches 0.

F. SERVER LOAD

FIGURE 11 compares the analytical result for all four caching strategies. FIFO caching performs the worst as $A \geq 0.7L$ is required for it to reach ≤ 0.05 server load. In contrast to this, random caching reaches it with $A \geq 0.5L$, while RLNC caching with $A \geq 0.25L$. Using the infinite caching, peers never need to download a packet from the server.

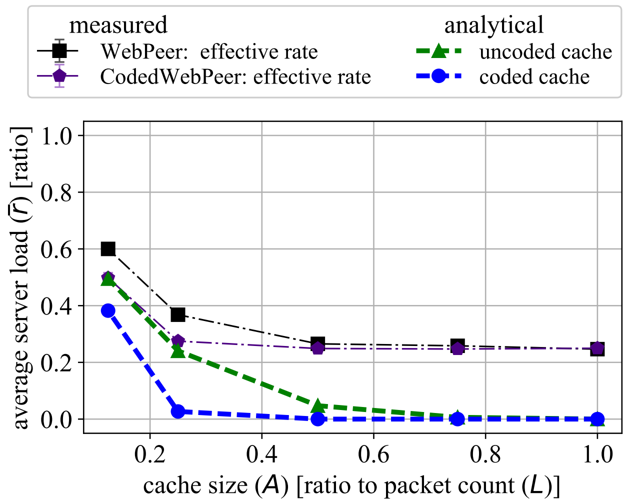

FIGURE 12. Calculated achievable server load r and r' compared to empirical effective server load with peers $N = 18$, generation size $g = 8$ and packets $L = 633$.

G. EFFECTIVE SERVER LOAD

As described in Section VI-C, our testbed can measure effective and achievable server load. In our analysis, we used the achievable server load to give a theoretical lower bound. We now compare r and r' to the measured effective server load in FIGURE 12.

The figure shows that *PasNet* behaves sub-optimally compared to the estimated achievable server load, because of the previously (Sec. VI-C) described technical challenges. By further optimizing the implementation of *PasNet*, the effective server load can be decreased. FIGURE 12 also presents that the random caching and the RLNC caching differ significantly at small cache sizes. This is an important result since, in practice, only a few percents of a full-length movie can be cached on mobile devices. Furthermore, by applying network coding, the effective server load of *PasNet* goes to r, the estimated achievable server load of the random caching strategy.

H. COMPARISON TO RELATED WORK

Comparing this to related research: Fijuta could reach n/α server upload capacity in a 2-hop network [14], where n is the number of peers, and α is the number of stripes that the video stream is divided to. This work differs from ours by the underlying network topology. While in our solution, peers can connect to any other peer, Fijuta focuses on multiple trees as the underlying topology of the overlay network. Furthermore, by analyzing large VoD vendors like BBC iPlayer [3] and Conviva [4], it has been shown that P2P-assisted VoD streaming has the potential to reduce the server load to 12%.

As 11 shows, depending on the cache size at the peers, with our solution the average server load approaches zero for large enough cache sizes. Using RLNC, the cache size can be significantly reduced compared to random solutions, while the server load stays close to zero.

VIII. CONCLUSION AND FUTURE WORK

For content providers to adopt P2P-assisted streaming solutions, a mathematical model is needed to predict the behavior (e.g., average server load) of the designed system. In this paper, we have presented an analytical framework for estimating the server load of mobile P2P-assisted streaming services. We have investigated four caching strategies: infinite, FIFO, random, and RLNC encoded caching. Using *PasNet*, our previously presented P2P system, we carried out extensive measurements to understand the behavior of a P2P system and to validate the accuracy of the framework at multiple steps of our calculations. Our results demonstrated that the average server load tends to zero with all caching methods as the cache size increases. By having only 5 P2P connections per peer and using the random caching, peers can store 40% fewer packets to achieve the same performance as with FIFO caching. Compared to the random caching, RLNC caching needs 50% fewer packets to achieve close-to-zero average server load. Furthermore, caching half of the packets, RLNC reaches zero server load, when the network is capable of serving all newcoming peers without the help of the server.

In its present state, our proposed framework can be used for extensive performance analysis. Our solution can be further extended to more closely match real-world scenarios by providing a detailed model of connection management and packet scheduling. Furthermore, we plan to investigate the QoE at the peers. We also intend to investigate different incentive mechanisms [27], [28] and incorporate them into our system to encourage peers to contribute their resources to the network.

Our work shows the potential of coded P2P-assisted streaming systems, which have high applicability in VoD and live streaming [29].

REFERENCES

[1] Ericsson. (Nov. 2018). *Ericsson Mobility Report*. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018

[2] (Oct. 2018). *The Global Internet Phenomena Report Sandive*. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report

[3] D. Karamshuk, N. Sastry, A. Secker, and J. Chandaria, "ISP-friendly peer-assisted on-demand streaming of long duration content in BBC iPlayer," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 289–297.

[4] A. Balachandran, V. Sekar, A. Akella, and S. Seshan, "Analyzing the potential benefits of CDN augmentation strategies for Internet video workloads," in *Proc. Conf. Internet Meas. Conf.*, New York, NY, USA, Oct. 2013, pp. 43–56, doi: 10.1145/2504730.2504743.

[5] P. J. Braun, M. Sipos, P. Ekler, and F. H. P. Fitzek, "On the performance boost for peer to peer WebRTC-based video streaming with network coding," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[6] P. J. Braun, P. Ekler, and F. Fitzek, "Network coding enhanced browser based peer-to-peer streaming," in *Proc. IEEE Int. Conf. Syst. Man, Cybern. (SMC)*, Budapest, Hungary, Oct. 2016, pp. 002104–002109.

[7] P. J. Braun, P. Ekler, and F. H. P. Fitzek, "Demonstration of a P2P assisted video streaming with WebRTC and network coding," in *Proc. 14th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2017, pp. 576–577.

[8] (2019). *Peer*. [Online]. Available: https://www.peer5.com

[9] Y. Chen, B. Zhang, Y. Liu, and W. Zhu, "Measurement and modeling of video watching time in a large-scale Internet video-on-demand system," *IEEE Trans. Multimedia*, vol. 15, no. 8, pp. 2087–2098, Dec. 2013.

[10] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 375–388, Oct. 2008, doi: 10.1145/1402946.1403001.

[11] W. Wu and J. C. S. Lui, "Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1492–1503, Aug. 2012.

[12] A. Shehab, M. Elhoseny, M. A. El Aziz, and A. E. Hassanien, "Efficient schemes for playout latency reduction in P2P-VoD systems," in *Advances in Soft Computing and Machine Learning in Image Processing*. Cham, Switzerland: Springer, Oct. 2018, pp. 477–495.

[13] G. Huang, L. Kong, K. Wu, and Z. Chen, "A service scheduling policy for improving playback quality of mesh-based P2P VoD systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. With Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, Dec. 2017, pp. 1311–1318.

[14] S. Fujita, "Cloud-assisted peer-to-peer video streaming with minimum latency," *IEICE Trans. Inf. Syst.*, vol. 102, no. 2, pp. 239–246, Feb. 2019.

[15] C. X. Mavromoustakis, G. Mastorakis, E. Pallis, C. Mysirlidis, T. Dagiuklas, I. Politis, C. Dobre, and K. Papanikolaou, "On the perceived quality evaluation of opportunistic Mobile P2P Scalable Video streaming," in *Proc. Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Aug. 2015, pp. 1515–1519.

[16] Y. Liu and S.-Z. Yu, "Efficient content delivery and caching scheme using network coding in CCN," in *Proc. Int. Conf. Comput. Netw. Commun. Technol. (CNCT)*. Paris, France: Atlantis Press, Nov. 2017, pp. 1–6. [Online]. Available: http://www.atlantis-press.com/php/paper-details.php?id=25870802

[17] P. J. Braun, M. Sipos, P. Ekler, and H. Charaf, "Increasing data distribution in BitTorrent networks by using network coding techniques," in *Proc. Eur. Wireless 21th Eur. Wireless Conf.*, May 2015, pp. 1–6.

[18] J. Miller, C. Gkantsidis, and P. Rodriguez, "Comprehensive view of a live network coding P2P system," in *Proc. IMC*, Oct. 2006, pp. 177–188. [Online]. Available: https://www.microsoft.com/en-us/research/publication/comprehensive-view-of-a-live-network-coding-p2p-system/

[19] P. Ekler, T. Lukovszki, and J. K. Nurminen, "Extending mobile BitTorrent environment with network coding," in *Proc. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2011, pp. 529–530.

[20] A. Azzalini, "A class of distributions which includes the normal ones," *Scand. J. Statist.*, vol. 12, no. 2, pp. 171–178, 1985.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. 18th Annu. Joint Conf. IEEE Comput. Commun. Societies Future Now (INFOCOM)*, vol. 1, Mar. 1999, pp. 126–134.

[22] T. T. Do, K. A. Hua, and M. A. Tantaoui, "P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment," in *Proc. IEEE Int. Conf. Commun.*, vol. 3, Jun. 2004, pp. 1467–1472.

[23] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and M. Médard, "On code parameters and coding vector representation for practical RLNC," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2011, pp. 1–5.

[24] B. Cohen. (Aug. 2016). *The BitTorrent Protocol Specification*. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[25] M. V. Pedersen, J. Heide, and F. H. P. Fitzek, "Kodo: An open and research oriented network coding library," in *Networking Workshops*, V. Casares-Giner, P. Manzoni, and A. Pont, Eds., Berlin, Germany: Springer, 2011, pp. 145–152.

[26] T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *Proc. 16th Int. Parallel Distrib. Process. Symp.*, Apr. 2002, p. 10.

[27] G. Huang, Y. Gao, L. Kong, and K. Wu, "An incentive scheme based on bitrate adaptation for cloud-assisted P2P video-on-demand streaming systems," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2018, pp. 404–408.

[28] S.-H. Lin, R. Pal, B.-C. Wang, and L. Golubchik, "On market-driven hybrid-P2P video streaming," *IEEE Trans. Multimedia*, vol. 19, no. 5, pp. 984–998, May 2017.

[29] N. Anjum, D. Karamshuk, M. Shikh-Bahaei, and N. Sastry, "Survey on peer-assisted content delivery networks," *Comput. Netw.*, vol. 116, pp. 79–95, Apr. 2017.

PATRIK J. BRAUN received the joint B.Sc. degree from the Budapest University of Technology and Economics (BME), Hungary, and Karlsruhe Institute of Technology, Germany, in 2013, the M.Sc. degree from BME, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Automation and Applied Informatics. He was a Guest Researcher as part of his Ph.D. degree with the Dresden University of Technology, Germany, during the summer of 2015, 2016, and 2017. He was a Fulbright Visiting Researcher with the Massachusetts Institute of Technology (MIT), from 2017 to 2018. His current research interests include networks, communication theory, and distributed caching.

ÁDÁM BUDAI received the M.Sc. degree in computer science from the Budapest University of Technology and Economics (BME), Hungary, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Automation and Applied Informatics. During his B.Sc., he was a Guest Researcher with the Chalmers University of Technology, Sweden, involved in the theory of plasma dynamics in tokamaks. His current research interests include medical image processing and reinforcement learning.

JÁNOS LEVENDOVSZKY received the Ph.D. degree from the Budapest University of Technology and Economics (BME) and the D.Sc. degree from the Hungarian Academy of Sciences. He is currently a full-time Professor with BME and also a Vice-Rector of science and innovation. His current research interests include adaptive signal processing, networking, artificial intelligence, and algebraic coding theory.

MÁRTON SIPOS received the B.Sc. and M.Sc. degrees in software engineering from the Budapest University of Technology and Economics (BME), in 2010 and 2012, respectively, and the joint Ph.D. degree from BME and Aalborg University, in 2018. His current research interest includes the development of erasure codes for distributed storage solutions. He has also been involved in the development and implementation of distributed storage clouds and mobile peer-to-peer storage solutions.

PÉTER EKLER received the Ph.D. degree from the Budapest University of Technology and Economics (BME), in 2011. He is currently an Associate Professor with the Department of Automation and Applied Informatics, BME. He has been involved in mobile Peer-to-Peer (P2P) and social networks for six years. He is the Creator of the first BitTorrent client for mainstream mobile phones based on Java ME platform. He was the coauthor of several mobile related scientific articles and book chapters. His current research interests include mobile-based social networks, P2P solutions, data analysis, and power law distributions in large networks. He has participated in several data warehouse and business intelligence related projects. He teaches mobile software development for several mobile platforms.

FRANK H. P. FITZEK received the Diploma (Dipl.-Ing.) degree in electrical engineering from the University of Technology-Rheinisch-Westfälische Technische Hochschule (RWTH), Aachen, Germany, in 1997, the Ph.D. (Dr.-Ing.) degree in electrical engineering from Technical University Berlin, Germany, in 2002, and the Honorary degree (Doctor Honoris Causa) from the Budapest University of Technology and Economy, in 2015. He was an Adjunct Professor with the University of Ferrara, Italy, in 2002. In 2003, he joined Aalborg University as an Associate Professor, where he became a Professor. He co-founded several start-up companies, starting with Acticom GmbH, Berlin, in 1999. He is currently a Professor and also the Head of the Deutsche Telekom Chair of Communication Networks, Technical University Dresden, Germany, where he coordinates the 5G Laboratory, Germany. His current research interests include wireless and mobile 5G communication networks, mobile phone programming, network coding, cross layer, and energy efficient protocol design and cooperative networking. He was a recipient of the NOKIA Champion Award several times in a row, from 2007 to 2011. In 2008, he received the Nokia Achievement Award for his work on cooperative networks. In 2011, he received the SAPERE AUDE Research Grant from the Danish Government. In 2012, he received the Vodafone Innovation prize.