AERO: Adaptive Emergency Request Optimization in CDN-P2P Live Streaming

João F. A. e Oliveira,* Ítalo Cunha,* Eliseu Miguel,+* and Sérgio V. A. Campos*
*Department of Computer Science — Universidade Federal de Minas Gerais (UFMG)
+Departamento de Ciência da Computação — Universidade Federal de Alfenas (UNIFAL-MG)
{holiver,cunha,scampos}@dcc.ufmg.br, eliseu.miguel@unifal-mg.edu.br

***Abstract*—Live streaming platforms employ advanced mechanisms to guarantee continuous and scalable video playback to large user bases. One such mechanism is CDN-P2P streaming, where servers are hosted on content distribution networks and client resources are used to help disseminate content in a peer-to-peer overlay. In CDN-P2P streaming, a peer that is about to miss the playback deadline of a content piece issues an emergency request to the CDN. Emergency requests allow the retrieval of nearly missed pieces and guarantee continuous playback. We show that emergency requests deliver a chunk close to its deadline and leave no time for dissemination through the peer-to-peer overlay, decreasing scalability. We present AERO, a mechanism that dynamically adjusts the rate at which CDN-hosted servers seed content pieces into the peer-to-peer overlay as a function of network conditions. Our evaluation of AERO under diverse conditions shows it reduces emergency requests, guarantees efficient peer-to-peer dissemination, and provides significant server upload bandwidth savings.**

I. Introduction

Peer-to-peer (P2P) content distribution improves the scalability of live streaming platforms and reduces costs. One disadvantage is that P2P distribution is best-effort and cannot guarantee quality of service (QoS). This is especially impactful in real-time live streaming [1]. Modern streaming platforms complement P2P distribution with reliable high-capacity servers hosted on content delivery networks (CDNs) [2]–[7]. To guarantee QoS, some of these platforms allow peers to issue 'emergency' requests to servers whenever the P2P overlay fails to distribute a video chunk before its playback deadline [2], [3], [8]–[11]. CDNs are well suited for such a solution as they are built to handle variable load and can guarantee QoS up to the contracted capacity.

Streaming cost in hybrid CDN-P2P streaming platforms is proportional to bandwidth utilization at the CDN-hosted servers. The challenge is to balance the number of seeded streams and emergency requests to minimize cost. We show that static allocation of seeded streams may become suboptimal depending on network conditions (Sec. IV).

In this paper we present Adaptive Emergency Request Optimization, AERO, a mechanism to minimize streaming cost (Sec. V). AERO dynamically computes the number of streams servers seed into the P2P overlay to reduce total bandwidth consumption on servers. When the P2P overlay is distributing video chunks efficiently and peers have unused upload bandwidth, AERO decreases the number of seeded streams. When the P2P overlay is not distributing video chunks efficiently but peers have unused upload bandwidth, AERO increases the number of seeded streams. The bandwidth consumption from the additional seeded streams is compensated by a greater reduction in the rate of emergency requests, as seeded peers start using spare upload bandwidth to redistribute video chunks through the P2P overlay.

AERO is flexible and can be used in combination with previous mechanisms for CDN-P2P live streaming. AERO is complementary to P2P overlay organization mechanisms [12]–[15]; in particular, AERO does not impose any structure on the P2P overlay and can be applied to both tree-like and mesh overlays. AERO is also complementary to peer discovery [16], chunk scheduling [17], and server placement algorithms [18].

Our evaluation shows that the rate of emergency requests is a good estimator for P2P distribution efficiency (Sec. VI). AERO monitors the rate of emergency requests to adapt the number of seeded streams to variable overlay conditions. We evaluate AERO across different scenarios varying peer upload bandwidth, fraction of free-riding peers, peer churn (up to flash crowds), overlay sizes, and overlay maintenance policies. Our results show that AERO increases server upload bandwidth savings by up to 30% compared to static configuration of the number of seeded streams.

II. CDN-P2P Live Streaming

Peer-assisted CDN (CDN-P2P) live streaming systems [2]–[8] have a back-end that captures, encodes, and splits a video stream into *chunks*. Chunks have fixed size but may contain multiple video frames under variable bit rate encoding. The back-end distributes video chunks to a set of well-provisioned *servers* hosted on CDNs or on cloud providers. Servers *seed*, i.e., immediately redistribute, video chunks received from the back-end to a subset of client *peers*. Peers collaborate with each other to receive and redistribute video chunks in a P2P overlay. Peers have a local buffer to store video chunks downloaded before their playback and to synchronize playback at a constant delay relative to the video's capture. Servers answer *emergency requests* that peers issue when the P2P overlay does not deliver a chunk before its playback deadline. Emergency requests ensure the majority of chunks are delivered and guarantee continuous playback. In CDN-P2P streaming, total cost to the content provider is dominated by the aggregate upload bandwidth across all servers.

Table I: Definitions and notation.

VAR.	DEFINITION
$\mathcal{P}$	Set of all peers.
$\mathcal{I}_p$	Set of input neighbors of peer p.
$\mathcal{O}_p$	Set of output neighbors of peer p.
$\lvert\mathcal{O}_p\rvert$	Out-degree of peer p, $\min(\lfloor B_p/R \rfloor, 10)$.
S	Server's aggregated seeding ratio.
$\mathcal{O}_S$	Set of seeded peers, number of seeded streams.
B_p	Upload bandwidth of peer p.
R	Stream encoding rate.
U_p	Peer p's upload rate (to neighbors in $\mathcal{O}_p$).
C_t	Server upload bandwidth consumption at round t.
E_t	Seeding ratio error at round t ($C_t - R\lvert\mathcal{O}_S\rvert_{t-1}$).
δ	AERO's seeding ratio scaling factor ($\delta \leq \Delta$).

We consider a CDN-P2P system with a *tracker* that maintains membership information for each streaming channel, i.e., which peers are receiving the stream and participating in its P2P overlay. Peers register with the tracker and query membership information to discover potential neighbors and join the P2P overlay. Peers periodically report neighborhood information and performance metrics to the tracker. The tracker aggregates reports to build snapshots of the P2P overlay and compute overlay properties.

Each peer p joins the streaming channel in *initialization mode*, where p requests chunks directly to CDN servers. This initialization reduces the time to fill peer p's buffer and ensures data availability. Initialization mode ends when peer p fills its buffer and establishes neighborhood partnerships with a subset of the peers it received from the tracker. Each peer p maintains two sets of neighbors. A peer p receives chunks from a set of *input neighbors*, denoted $\mathcal{I}_p$, and sends chunks to a set of *output neighbors*, denoted $\mathcal{O}_p$. We call the number of input and output neighbors, i.e., $|\mathcal{I}_p|$ and $|\mathcal{O}_p|$, a peer's *in-* and *out-degree*, respectively. Tab. I summarizes notation.

If a peer p is not receiving all the chunks it needs from its input neighbors (i.e., is issuing emergency requests) it will remove from $\mathcal{I}_p$ the neighbor with highest number of unanswered chunk requests. A peer p limits its out-degree, $|\mathcal{O}_p|$ to the number of streams it can upload (i.e., the ratio between the peer's upload bandwidth and the video streaming rate) and denies partnership requests if $\mathcal{O}_p$ is full. A peer p periodically refreshes its list of known peers by contacting the tracker and by exchanging membership information with other known peers [16].

Peers periodically send buffer maps to output neighbors in $\mathcal{O}_p$ announcing chunks stored in their buffer. Peers generate chunk requests sequentially, i.e., earliest deadline first, and serve requests in order of arrival. When multiple neighbors can provide video chunks, a peer requests a number of chunks proportional to each neighbor's upload bandwidth.

The benefit in P2P overlay distribution obtained from seeding the stream to a peer is proportional to that peers's upload bandwidth [15], [19], [20]. As a result, servers choose seeded peers (i.e., peers receiving chunks directly from CDN servers) by decreasing upload bandwidth and computing the number of seeded peers as a function of the obtained benefit. Servers seed streams to a set of peers denoted $\mathcal{O}_S$. The number of seeded streams, $|\mathcal{O}_S|$, is defined as a function of the *seeding ratio* S, the ratio between the aggregate upload bandwidth of seeded peers and the aggregate upload bandwidth of all peers. More formally, if $\mathcal{P}$ is the set of all peers, B_p is the upload bandwidth of peer p, then the number of seeded peers is:

$$|\mathcal{O}_S| = \min\left\{|\mathcal{S}| \;\middle|\; \mathcal{S} \subseteq \mathcal{P} \wedge \sum_{p\in\mathcal{S}} B_p \geq S \sum_{q\in\mathcal{P}} B_q\right\}. \quad (1)$$

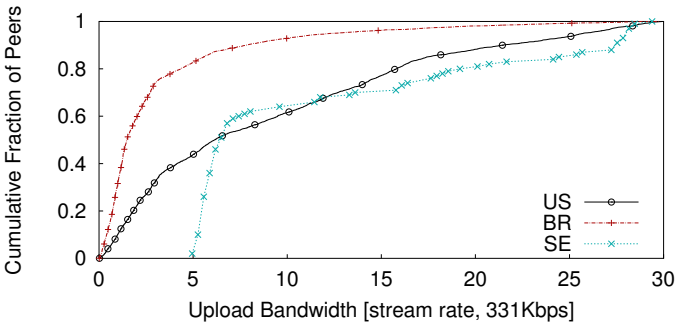


Figure 1: Peer upload bandwidth distributions.

III. Simulation Setup

In this section we describe the simulation environment and experiment setup we use to evaluate distribution efficiency in Secs. IV and VI. We evaluate CDN-P2P streaming costs using a simulator which shares most of its code with a real implementation of the CDN-P2P system described in Sec. II. The simulator substitutes the system's networking code for a network emulator that allows control of a number of characteristics such as link bandwidths, transmission delays, queueing, and NATs. We repeat each simulation five times. Numerical results ignore data from the first three minutes of all simulations to avoid the overlay's warm-up period.

Underlay configuration. We configure the network emulator without any underlay bandwidth constraints (but we do limit peer upload bandwidth below) and set the transmission error rate to zero. We choose the one-way latency between peers uniformly distributed between 10ms and 50ms. Although the underlay never drops video chunks, we note the P2P overlay may still fail to disseminate a video chunk due to lack of peer upload bandwidth and inefficient neighborhoods leading to long, high-latency dissemination paths.

Peer upload bandwidth distributions. We evaluate CDN-P2P streaming costs using the three different peer upload bandwidth distributions shown in Fig. 1. Upload bandwidth distribution for North American ('US' line) and Brazilian ('BR' line) hosts were collected from TestMy.net, an online bandwidth test service. TestMy.net offers reports of the last 10000 upload and download bandwidth measurement performed on its servers from each country. We filtered upload bandwidths greater than 10Mbps to avoid distortions caused by potentially non-residential connections. The last distribution ('SE' line) was collected from upload bandwidths observed in an operational test deployment with thousands of peers over a Swedish corporate network. From each distribution, we generate four resource-constrained scenarios. We consider

two scenarios where we divide peer bandwidths by two and four, denoted DIV2 and DIV4, respectively. This is equivalent to multiplying the streaming rate by two and four, e.g., as needed for streaming HQ videos. We also consider scenarios where 50% and 75% of peers are free-riders, i.e, do not upload video chunks, denoted 50F and 75F, respectively. Note that scenarios DIV2 and 50F have equivalent average peer upload bandwidth; DIV4 and 75F also have equivalent average peer upload bandwidth. Unless stated otherwise, we use the bandwidth distribution of North American hosts as a baseline and refer to it as BASE.

Overlay sizes. We evaluate channel populations varying from 100 to 2000 simultaneous peers. Peers join the overlay at a constant rate over the first 100 seconds of the simulation. Unless stated otherwise, we run simulations with 500 peers.

Overlay configuration. We assume servers have bandwidth to upload the number of streams required to achieve the configured seeding ratio S, to support peers in initialization mode, and answer all emergency requests. We set the seeding ratio $S = 2.5\%$. This seeding ratio has been observed to yield positive results in previous work and is small enough to allow cost savings. We vary peer in-degrees $|\mathcal{I}_p|$ between 2 and 5, and maximum peer out-degrees $|\mathcal{O}_p|$ between 0 (e.g., as in free-riders) and 10. Unless stated otherwise, we set $|\mathcal{I}_p| = 3$ and $|\mathcal{O}_p| = \min(\lfloor B_p/R \rfloor, 10)$, where B_p is peer p's upload bandwidth and R is the streaming rate. The out-degree is bounded by the number of streams a peer can upload to avoid receiving more requests than can be served and by a fixed threshold to limit abuse of peer upload capacity.

Peer behavior. We emulate peer churn and flash crowds to further stress the P2P overlay. We consider a pessimistic scenario for peer churn where we remove and insert 3% of peers in the overlay at random every ten seconds. This churn rate is at least three times higher than previously found in other studies [8], [18]. We emulate flash crowds alternating between adding and removing 1000 peers from an overlay with 100 peers every ten minutes.

IV. P2P Distribution Efficiency

Our goal is to evaluate P2P distribution efficiency for different network configurations. We show that distribution efficiency can degrade significantly for bandwidth-constrained scenarios even though peers have spare upload bandwidth.

Video chunks are either distributed by the server (seeding and emergency requests) or disseminated by the P2P overlay. We note that peers that do not receive a video chunk from the P2P overlay issue an emergency request for that chunk.

We measure P2P distribution efficiency as the fraction of chunks that servers did not have to distribute due to the P2P overlay, which we refer to as *savings*. More precisely, savings is the ratio between the number of chunks distributed by the P2P overlay and the total number of distributed chunks.

Fig. 2 shows savings over time for four different peer upload bandwidth scenarios. Results for DIV2 (omitted) are quantitatively similar to results for 50F. Savings for BASE and 50F stabilize at 96% and 95% approximately three minutes after the start of the experiment. In these scenarios, seeding, peer initialization, and emergency requests account for 4–5% of all distributed chunks. Savings increases sharply in the beginning as peers optimize the overlay self-organizing away from servers by decreasing upload bandwidth. Fig. 2 shows that savings for the 75F and DIV4 scenarios stabilize substantially lower at 77% and 50%, respectively. This means that servers are responsible for distributing 23% and 50% of all chunks, workloads that are 5 and 10 times higher than in the BASE scenario. One may think that savings is low because peer upload bandwidth is fully utilized and insufficient to distribute chunks, but we find this is not the case. We identify three reasons for inefficient P2P distribution.

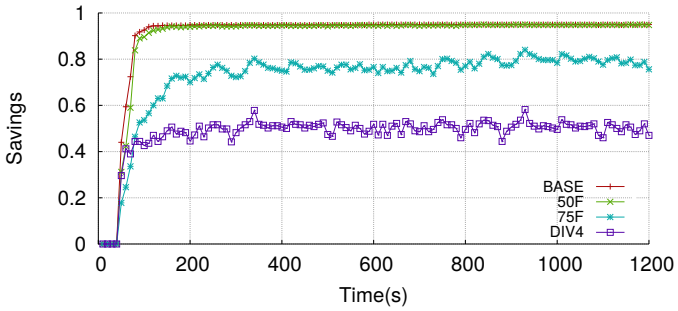


Figure 2: Savings for different upload bandwidth distributions.

Table II: Origin of video chunks received.

CHUNK ORIGIN	BASE	50F	75F	DIV4
Seeded by servers	1.4%	0.8%	0.3%	0.7%
Emergency request	3.0%	4.9%	25.9%	49.5%
P2P overlay	95.6%	94.3%	73.8%	49.8%

Peers cannot establish input partnerships. Low upload bandwidths lead to low out-degrees, $|\mathcal{O}_p|$. As each peer tries to establish input partnerships with peers that have equivalent or more upload bandwidth than itself, low out-degrees lead to competition for output partnerships between high-bandwidth peers and compromise overlay stability.

Peers underutilize upload bandwidth. We have studied the effective upload bandwidth utilization over time for different upload bandwidth distributions. Effective upload bandwidth utilization is the rate at which a peer p uploaded chunks to its output neighbors, U_p, divided by the maximum rate at which that peer could upload chunks, $\min(B_p, |\mathcal{O}_p|R)$. The fraction of peers that cannot seed any stream and have zero output neighbors (i.e., $|\mathcal{O}_p| = 0$) is 10%, 55%, 77%, and 39% in BASE, 50F, 75F, and DIV4, respectively. Peers have average utilizations of 17% and 32% in the BASE and 50F scenarios, respectively, which is enough to redistribute most chunks and achieve high savings (Fig. 2). Peers in the 75F and DIV4 scenarios also have spare upload bandwidth capacity; savings, however, remains low.

Emergency requests are not P2P-friendly. Chunks obtained through emergency requests have short lifespans in the P2P overlay and cannot be widely distributed. Tab. II shows, for different upload bandwidth distributions, the average fraction of chunks seeded by servers, transmitted as answers to emergency requests, and redistributed by the P2P overlay. We

Table III: Average number of retransmisions of a video chunk after it enters the P2P overlay.

CHUNK ORIGIN	BASE	50F	75F	DIV4
Seeded	2.65	3.07	3.59	0.84
Emergency	0.01	0.30	0.47	0.43

observe that bandwidth-constrained scenarios result in significantly higher fractions of emergency requests than BASE. Tab. III shows the average number of retransmissions for chunks seeded by servers and for chunks sent as answers to emergency requests. Seeded requests are transmitted early, stay longer in the P2P overlay, and are retransmitted more times than chunks from emergency requests. Peers that issue emergency requests may fail to redistribute chunks, leading to their output neighbors also issuing emergency requests, degrading overall P2P distribution efficiency. This gets more evident as resources get more constrained.

V. Adaptive Emergency Request Optimization

We propose AERO, the Adaptive Emergency Request Optimization mechanism. AERO's goal is to minimize server bandwidth consumption by increasing P2P distribution efficiency. Our main idea is to configure the seeding ratio S dynamically. AERO increases the seeding ratio when P2P distribution efficiency is low, seeding chunks to peers that would otherwise receive most chunks via emergency requests. These peers can then use their upload bandwidth more effectively and possibly start new chunk redistribution chains in the P2P overlay. AERO decreases the number of seeded streams when P2P distribution efficiency stabilizes in an attempt to maximize savings. AERO's goal is to track a practical seeding ratio as network conditions and P2P overlays change.

AERO configures the seeding ratio S as a function of server upload bandwidth consumption due to seeding and emergency requests. AERO does not consider upload bandwidth used to support peers in initialization mode because this bandwidth demand is fixed per peer and because peer arrival times are outside the control of servers.

AERO operates in rounds. Each round, AERO decides whether to increase or decrease the seeding ratio S depending on historical seeding ratio allocations and bandwidth consumption observations, as shown in Alg. 1.

Whenever bandwidth consumption is not stable, AERO updates the seeding ratio S according to the seeding ratio error E_t and E_{t-1} of the last two rounds. The seeding ratio error at round t, E_t, is defined as the difference between the total server upload bandwidth consumption, C_t, and the expected bandwidth consumption, $R|\mathcal{O}_S|_{t-1}$. The seeding ratio error is positive when the seeding ratio is low and servers receive a large number of emergency requests (underprovisioning). Conversely, the seeding ratio error is negative when the seeding ratio is high and seeded peers receive chunks from the P2P overlay (overprovision).

If $\text{sign}(E_t) = \text{sign}(E_{t-1})$, the last two rounds overestimated or underestimated the optimal seeding ratio. AERO concludes it is not close to the optimum seeding ratio and increases the scaling factor by a constant factor (shown as 0.75 in Alg. 1) to improve convergence speed. If $\text{sign}(E_t) \neq \text{sign}(E_{t-1})$, then one round underestimated and the other overestimated the optimal seeding ratio. AERO decreases the scaling factor by a constant factor (shown as 0.75 in Alg. 1) to converge closer to the optimal seeding ratio. The scaling factor is restored to its upper bound whenever a jump in bandwidth consumption is detected, i.e., whenever bandwidth consumption doubles or halves between rounds. At each round, AERO increases the seeding ratio by the scaling factor when the system is underprovisioned ($E_t > 0$) and decreases the seeding ratio when the system is overprovisioned ($E_t < 0$).

Algorithm 1: AERO's algorithm to update the seeding ratio S at each round

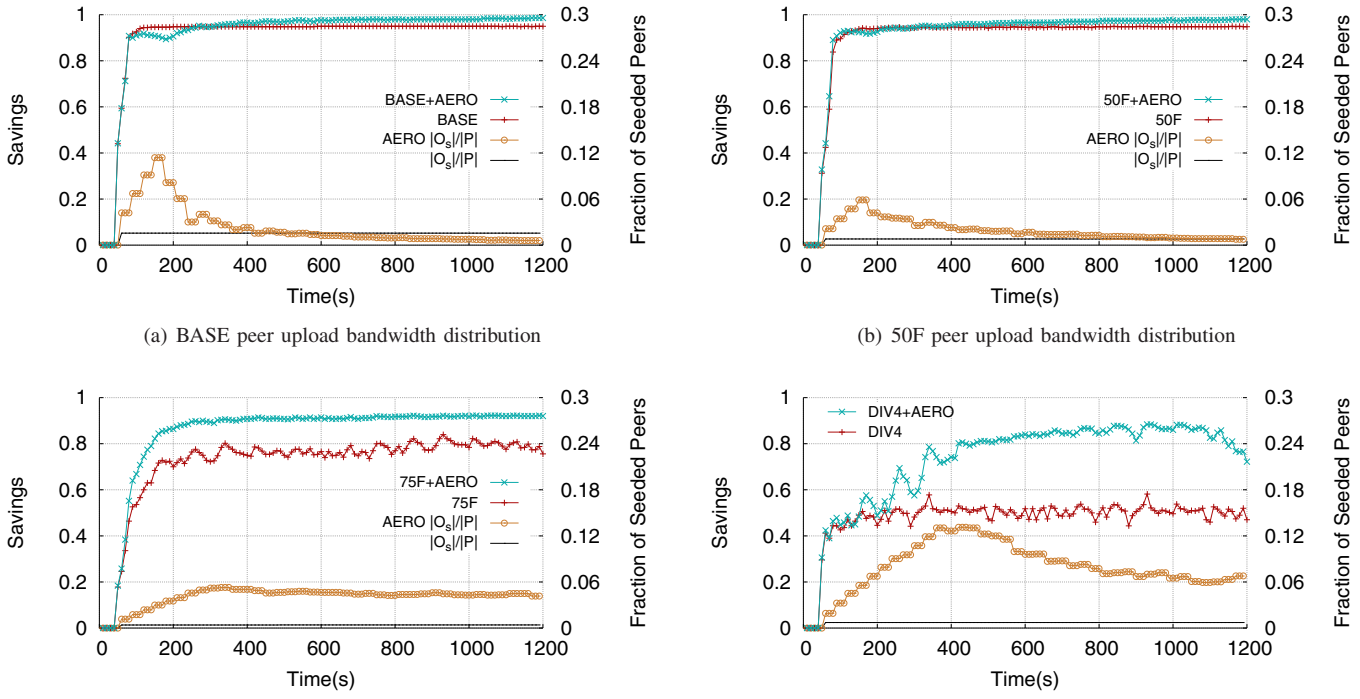
```
input: history of bandwidth consumption, seeding ratio
input: seeding ratio scaling factor δ (upper bound Δ)
input: seeding ratio error E at rounds t and t − 1
if bandwidth consumption is stable then S ← 0.95S;
else
    if bandwidth consumption jump then δ ← Δ;
    else
        if sign(E_t) = sign(E_{t−1}) then
            δ ← min(δ/0.75, Δ);
        else δ ← 0.75δ;
    end
    S ← S + sign(E_t)δ
end
```

The process above iterates until the scaling factor decreases enough that server bandwidth consumption stabilizes. To avoid getting stuck at a local minimum, AERO systematically decreases the seeding ratio by a constant factor once bandwidth consumption stabilizes (shown as 0.95 in Alg. 1). This systematic reduction of the seeding ratio will either reduce server bandwidth consumption or cause fluctuations due to underprovisioning. When bandwidth fluctuates, AERO will reconverge to an optimum seeding ratio (possibly the same).

AERO keeps a history of bandwidth consumption at CDN servers to compute whether bandwidth consumption is stable or not. AERO considers that bandwidth consumption is increasing or decreasing if current bandwidth consumption is higher or lower than those observed in the last three rounds. If the current bandwidth consumption is similar to those observed in previous rounds or if a trend is unclear, AERO considers that bandwidth consumption is stable. More precisely, the trend of bandwidth consumption is given by the following, where $I(\cdot)$ in an indicator function that returns 1 if its parameter is true and zero otherwise.

$$\text{trend} = \begin{cases} \text{increasing} & \text{if } \sum_{i=1}^{3} I(0.95C_t > C_{t-i}) > 1, \\ \text{decreasing} & \text{if } \sum_{i=1}^{3} I(1.05C_t < C_{t-i}) > 1, \\ \text{stable} & \text{otherwise.} \end{cases} \quad (2)$$

Stability periods may end due to varying network and P2P overlay conditions, e.g., peer churn or flash crowds. AERO's round duration must be long enough to allow AERO to



(a) BASE peer upload bandwidth distribution

(b) 50F peer upload bandwidth distribution

(c) 75F peer upload bandwidth distribution

(d) DIV4 peer upload bandwidth distribution

Figure 3: Comparison of overall savings between scenarios with and without AERO.

observe the impact of seeding ratio adaptation decisions on bandwidth consumption. This property is necessary to compute the seeding ratio errors E_t. AERO makes no assumptions about overlay properties or peer behavior, and can be deployed in conjunction with existing overlay maintenance or chunk scheduling mechanisms.

VI. Evaluation

We have evaluated AERO for different deployment scenarios varying peer upload bandwidth distributions, overlay construction policies, overlay sizes, and peer behavior. As in Sec. IV, we consider four out of five different BASE peer upload bandwidth distribution derivatives: BASE, 50F, 75F, and DIV4. We show that AERO increases P2P savings across all scenarios, trading bandwidth consumption spent on emergency requests for seeding video streams, improving P2P distribution efficiency.

Unless stated otherwise, we use the default parameters defined in Sec. III and use AERO with 30-second rounds, initial seeding ratio $S = 2.5\%$, maximum seeding ratio modifier $\Delta = 5\%$.

A. Peer upload bandwidth

Fig. 3 compares server bandwidth savings for different peer upload bandwidth distributions with and without AERO on streaming channels with 500 peers (left y-axes). The lines labeled BASE, 50F, 75F, and DIV4 are identical to those in Fig. 2. Figs. 3(a) and 3(b) show that AERO adapts to high-resource scenarios and further increases savings. Although absolute difference is small, AERO reduces average server upload bandwidth by up to 43% (relative difference between using and not using AERO). Figs. 3(c) and 3(d) show that AERO achieves even higher savings in resource-constrained scenarios (and up to 60% reduction of average server upload bandwidth). We note that there are no "missed" chunks; both solutions (with and without AERO) distribute the same number of chunks. AERO's savings come entirely from higher P2P distribution efficiency.

To explain this behavior, Fig. 3 also shows the fraction of seeded peers configured by AERO and the fraction of seeded peers statically configured when not using AERO (right y-axes, with range up to 30%). In high-resource scenarios (Figs. 3(a) and 3(b)), AERO reduces the fraction of seeded peers while maintaining P2P distribution efficiency. AERO achieves higher bandwidth savings when its fraction of seeded peers gets lower than in the static configuration. In resource-constrained scenarios (Figs. 3(c) and 3(d)), we observe AERO increases the number of seeded streams compared to the static configuration. This allows peers to distribute more chunks through the P2P overlay and helps further reduce the number of emergency requests. Finally, we note that AERO seeds to a lower fraction of peers in scenario 75F compared to DIV4. On average, high-resource peers in 75F have four times more bandwidth than high-resource peers in DIV4, and less seeding is necessary to achieve efficient P2P distribution. Tab. IV

Table IV: Origin of received video chunks when using AERO.

CHUNK ORIGIN	BASE	50F	75F	DIV4
Seeded by servers	1.6%	1.6%	3.2%	6.6%
Emergency request	0.9%	1.6%	5.9%	13.4%
P2P overlay	97.5%	96.8%	90.9%	80.0%

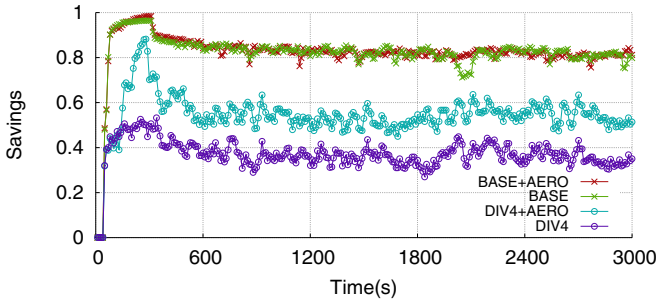


Figure 4: Savings under peer churn.

shows the fraction of chunks that were distributed by seeding, emergency requests, and the P2P overlay. A comparison with Tab. II shows AERO significantly improves P2P efficiency.

B. Peer churn and flash crowds

We also evaluate AERO under adverse peer behavior. Fig. 4 shows savings for the BASE and DIV4 scenarios when we turn on peer churn at second 300. Churn replaces peers that have complete neighborhoods for new peers that have empty buffers and zero partnerships, disrupting the P2P overlay and further degrading savings. Churn can be especially harmful if it removes a significant fraction of seeded peers from the overlay at once. We have manually verified one such case in one simulation run for the BASE scenario at around 2000 seconds, which can be seen in Fig. 4 even after averaging all five simulation runs. Fig. 4 shows that AERO's dynamic seeding ratio configuration adapts to very dynamic overlays and provides equivalent or higher bandwidth savings compared to static seeding ratios (results for 75F are omitted but qualitatively similar to DIV4).

Fig. 5 shows savings over time for the BASE and DIV4 scenarios when the streaming channel experiences periodic flash crowd events. We start with an overlay containing 100 peers, and alternate between adding and removing 1000 peers from the overlay periodically every 10 minutes. Peers join

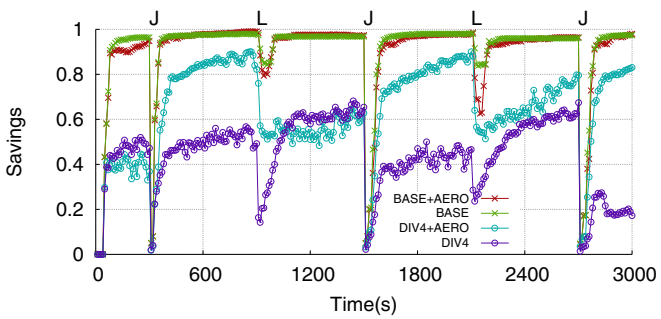


Figure 5: Savings over time for periodic flash crowd events.

the overlay at seconds 300, 1500, and 2700, denoted 'J', in initialization mode and significantly degrade savings, as shown by the simultaneous valleys. AERO then quickly adjusts the seeding ratio and achieves high savings. Peers leave at seconds 900 and 2100, denoted 'L', severing the majority of partnerships and leaving a (possibly disconnected) P2P overlay with 100 peers. After both arrival and departure events, AERO achieves equivalent or higher savings compared to static configuration of seeding ratios.

VII. Related Work

Efficient P2P live streaming. A large body of work has been dedicated to building and evaluating algorithms and mechanisms for efficient P2P live streaming. Contributions cover techniques to optimize P2P overlay topologies [18]–[21], schedule chunk requests and transmissions [17], [21], adapt topologies according to overlay and network conditions [22], increase peer cooperation and avoid free-riding [23]–[25], mitigate collusion attacks [24], handle peer churn and flash crowds [26].

These works optimize a P2P overlay and may be applicable to CDN-P2P systems. However, they do not consider emergency requests, a central point in AERO's design. We argue AERO can be used in conjunction with these techniques to jointly optimize CDN-server seeding ratios and P2P distribution. We note our CDN-P2P streaming system incorporates findings from previous work, e.g., it organizes peers away from servers by decreasing upload bandwidth [15], [19], [20].

Efficient CDN live streaming. Researchers have already studied the use of CDNs to deploy live streaming systems. An example is VDN, which proposes control algorithms for reliable streaming of large scale events using CDNs while reducing costs [27]. These techniques, however, do not consider P2P distribution and are mostly orthogonal to AERO and our contributions. In this work, we rely on these techniques when we assume that CDN edge servers receive all video chunks and have enough bandwidth to seed peers and answer all emergency requests.

CDN-P2P live streaming. Researchers have also studied how to use system parameters and performance metrics—such as user churn rate, stream rate, average user upload bandwidth, maximum tolerable chunk distribution delay—to build resource allocation models for hybrid CDN-P2P systems. These allocation models can be used, e.g., to provision upload bandwidth at streaming servers [28], deciding where to connect new peers joining the stream (e.g., CDN servers or the P2P overlay) [6], or to switch between CDN and P2P operation [7].

AERO is similar, as computing the seeding ratio is a form of resource allocation. These solutions, however, do not consider emergency requests and are not directly comparable to AERO. Compared to these solutions, AERO does not require detailed information about system parameters, network conditions, and performance metrics. AERO is a simpler alternative that can efficiently allocate resources on demand in diverse scenarios.

Emergency requests and quality of service. Some P2P streaming systems have proposed mechanisms equivalent to emergency requests to guarantee quality of service [8]–[10]. Although these works are related, they consider different goals and different deployment scenarios than AERO. For example, CLive [8] is similar to our work as it adds cloud-hosted servers as seeding supernodes in the P2P overlay. CLive's goal is to guarantee QoS by limiting chunk delivery delay and the rate of missed chunks, different from AERO's goal of minimizing streaming costs. Moreover, while CLive estimates demand based on P2P overlay properties to provision cloud servers, AERO is oblivious to overlay properties other than the resulting emergency requests. Compared to CLive, AERO requires less monitoring overhead and can be more easily deployed as it makes decisions based only on server bandwidth consumption, which is trivial to obtain.

VIII. Conclusion

CDN-P2P live streaming systems allow clients to issue emergency requests and guarantees QoS. In this paper, we have shown that emergency requests deliver chunks close to their playback deadlines and allow only a short period of time for chunk redistribution in the P2P overlay. This leads to decreased P2P distribution efficiency.

We have presented AERO, a mechanism to minimize bandwidth consumption in CDN-P2P live streaming systems. AERO first increases the amount of video streams seeded into the overlay to guarantee that the overlay is effectively using peer upload bandwidth to redistribute chunks, reducing emergency requests. AERO then attempts to reduce bandwidth consumption at servers reducing the amount of seeded video streams without increasing the rate of emergency requests. Our evaluation of AERO under diverse conditions shows that it achieves up to 60% higher server upload bandwidth savings compared to static allocation of the number of seeded streams.

AERO requires readily available information: the number of video streams CDN servers are seeding into the P2P overlay and the number of emergency requests. AERO runs entirely on CDN servers and does not require modification of client software. AERO also does not impose any restriction on and is compatible with existing P2P overlay construction and chunk scheduling mechanisms. We believe AERO can be easily integrated into existing systems that use emergency requests or similar mechanisms to guarantee quality of service, helping improve scalability and reducing operational costs.

Acknowledgments

This work is partly funded by FAPEMIG, CAPES, and CNPq.

References

[1] Y. Lu, B. Fallica, F. Kuipers, R. Kooij, and P. Van Mieghem, "Assessing the Quality of Experience of SopCast," *IJIPT*, vol. 4, no. 1, pp. 11–23, 2009.

[2] R. Roverso, R. Reale, S. El-Ansary, and S. Haridi, "SmoothCache 2.0: CDN-quality Adaptive HTTP Live Streaming on Peer-to-peer Overlays," in *Proc. ACM MMSys*, 2015.

[3] R. Roverso, S. El-Ansary, and S. Haridi, "Smoothcache: Http-Live Streaming goes Peer-to-Peer," in *Proc. IFIP Networking*, 2012.

[4] Z. Lu, Y. Wang, and Y. Yang, "An Analysis and Comparison of CDN-P2P-Hybrid Content Delivery System and Model," *J. of Communications*, vol. 7, no. 3, pp. 232–245, 2012.

[5] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec, "Peer-assisted Content Distribution in Akamai NetSession," in *Proc ACM IMC*, 2013.

[6] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky," in *Proc. ACM Multimedia*, 2009.

[7] A. Mansy and M. Ammar, "Analysis of Adaptive Streaming for Hybrid CDN/P2P Live Video Systems," in *Proc. IEEE ICNP*, 2011.

[8] A. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi, "CLive: Cloud-assisted P2P Live Streaming," in *Proc. IEEE P2P*, 2012.

[9] G. Kreitz and F. Niemela, "Spotify – Large Scale, Low Latency, P2P Music-on-Demand Streaming," in *Proc. IEEE P2P*, 2010.

[10] L. Zhang, "Efficient Video Streaming in Peer-to-Peer Networks," Ph.D. dissertation, The Hong Kong Polytechnic University, 2005.

[11] Y. Jin, Y. Yi, G. Kesidis, F. Kocak, and J. Shin, "Hybrid Client-Server and Peer-to-Peer Caching Systems with Selfish Peers," in *Proc. IEEE INFOCOM*, 2013.

[12] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. ACM SIGOPS*, 2003.

[13] X. Zhang, J. Liu, B. Li, and T. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network For Efficient Live Media Streaming," in *Proc. IEEE INFOCOM*, 2005.

[14] B. Li, S. Xie, Y. Qu, G. Keung, C. Lin, J. Liu, and X. Zhang, "Inside The New Coolstreaming: Principles, Measurements And Performance Implications," in *Proc. IEEE INFOCOM*, 2008.

[15] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia, "Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–1, 2014.

[16] R. Roverso, J. Dowling, and M. Jelasity, "Through the Wormhole: Low Cost, Fresh Peer Sampling for the Internet," in *Proc. IEEE P2P*, 2013.

[17] B. Zhao, J. Lui, and D. Chiu, "Exploring the Optimal Chunk Selection Policy for Data-Driven P2P Streaming Systems," in *Proc. IEEE P2P*, 2009.

[18] G. Simoni, R. Roverso, and A. Montresor, "RankSlicing: A Decentralized Protocol for Supernode Selection," in *Proc. IEEE P2P*, 2014.

[19] P. Felber and E. Biersack, "Cooperative content distribution: Scalability through self-organization," in *Self-star Properties in Complex Information Systems*. Springer, 2005, pp. 343–357.

[20] A. Payberah, J. Dowling, F. Rahimian, and S. H., "Distributed Optimization of P2P Live Streaming Overlays," *Special Issue on Extreme Distributed Systems: From Large Scale to Complexity*, vol. 94, no. 8, pp. 621–647, 2012.

[21] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "QoE in Pull based P2P-TV Systems: Overlay Topology Design Tradeoffs," in *Proc. IEEE P2P*, 2010.

[22] M. Wichtlhuber, B. Richerzhagen, J. Ruckert, and D. Hausheer, "TRANSIT: Supporting Transitions in Peer-to-Peer Live Video Streaming," in *Proc. IFIP Networking*, 2014.

[23] R. Guerraoui, K. Huguenin, A. Kermarrec, M. Monod, and S. Prusty, "LiFTinG: Lightweight Freerider-Tracking in Gossip," in *Proc. ACM/IFIP/USENIX Middleware*, 2010.

[24] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, "Contracts: Practical Contribution Incentives For P2P Live Streaming," in *Proc. USENIX NSDI*, 2010.

[25] G. Gonçalves, I. Cunha, A. Vieira, and J. Almeida, "Predicting the level of cooperation in a peer-to-peer live streaming application," *Multimedia Systems*, pp. 1–20, 2014.

[26] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, "Flash Crowd in P2P Live Streaming Systems: Fundamental Characteristics and Design Implications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 7, pp. 1227–1239, 2012.

[27] M. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, Real-time Centralized Control for CDN-based Live Video Delivery," in *Proc. ACM SIGCOMM*, 2015.

[28] S. Tewari and S. Menon, "On Resource Provisioning in Hybrid Peer-to-Peer Live Streaming Systems," in *Proc. IEEE BMSB*, 2009.