PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming

Simon da Silva, Sonia Ben Mokhtar, Stefan Contiu, Daniel Négru, Laurent Réveillère, Etienne Rivière

PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming

Simon Da Silva
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France

Sonia Ben Mokhtar
INSA Lyon, LIRIS, CNRS, France

Stefan Contiu
Scille SAS, France
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France

Daniel Négru
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France

Laurent Réveillère
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France

Etienne Rivière
ICTEAM, UCLouvain, Belgium

Abstract

Video on Demand (VoD) streaming is the largest source of Internet traffic. Efficient and scalable VoD requires Content Delivery Networks (CDNs) whose cost are prohibitive for many providers. An alternative is to cache and serve video content using end-users devices. Direct connections between these devices complement the resources of core VoD servers with an edge-assisted collaborative CDN.

VoD access histories can reveal critical personal information, and centralized VoD solutions are notorious for exploiting personal data. Hiding the interests of users from servers and edge-assisting devices is necessary for a new generation of privacy-preserving VoD services.

We introduce PrivaTube, a scalable and cost-effective VoD solution. PrivaTube aggregates video content from multiple servers and edge peers to offer a high Quality of Experience (QoE) for its users. It enables privacy preservation at all levels of the content distribution process. It leverages Trusted Execution Environments (TEEs) at servers and clients, and obfuscates access patterns using fake requests that reduce the risk of personal information leaks. Fake requests are further leveraged to implement proactive provisioning and improve QoE. Our evaluation of a complete prototype shows that PrivaTube reduces the load on servers and increases QoE while providing strong privacy guarantees.

CCS Concepts • **Security and privacy**;

Keywords multimedia, privacy, security, streaming, TEE

ACM Reference Format:
Simon Da Silva, Sonia Ben Mokhtar, Stefan Contiu, Daniel Négru, Laurent Réveillère, and Etienne Rivière. 2019. PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming. In *Middleware '19: Middleware '19: 20th International Middleware Conference, December 8–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3361525.3361546

Middleware '19, December 8–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7009-7/19/12…$15.00
https://doi.org/10.1145/3361525.3361546

1 Introduction

Video streaming accounted for close to 58% of *all* Internet traffic in 2018 [1] and is expected to reach 81% in 2021 [2]. There are two reasons for this success: the multiplication of video sources (e.g., video streaming catalogs, online TV channels, personal videos sharing) and the pervasiveness of high-quality Internet connections.

Dominating Video on Demand (VoD) platforms rely on large-scale infrastructures to cope with an increasing demand for high Quality of Experience (QoE) and high-bitrate content. YouTube, Netflix or Twitch have set up planetary-scale *proprietary* Content Delivery Networks (CDNs) [3, 4, 9, 23]. They further deploy extra CDN nodes directly at Internet Service Providers (ISPs) and negotiate special peering relations with their Autonomous Systems (ASs) [38]. Other platforms can rely on existing third-party CDNs to serve content. DailyMotion is reported to use the CDNs of Orange, Akamai, and Limelight to scale video delivery in different parts of the world [8].

Dedicated CDNs require a high up-front investment, and third-party CDNs incur high operational costs. The use of an *edge-assisted* CDN is an appealing alternative for smaller players or platforms that do not wish to monetize their users' personal data to sustain their activity, such as the free and open PeerTube [46] network. Edge-assisted CDNs complement core dedicated servers with the direct exchange of video content between end-users' devices. Examples of platforms using an edge-assisted CDN are LiveSky [62], Peer5 [45], Quanteec [49], Streamroot [54] and Kankan [64].

The usage of VoD platforms generates sensitive personal data in the form of access histories to videos. This data can be leveraged for the benefit of the user, e.g. allowing personalized recommendations for new videos, or for the benefit of the platform, e.g. for targeted advertising. However, the availability of access histories also leads to major threats to privacy. Indeed, this data can be used to infer private information about the user, such as her/his gender, origin, political, religious or sexual orientation. Kandias *et al.* [31] show for instance that the political affiliation of YouTube users can easily be extracted from their access histories. Luo *et al.* [36] similarly show how a household composition can be inferred.

Protecting users' privacy in a video streaming system requires hiding their access histories from servers and other users. Anonymizing networks such as TOR [24] allow hiding the identity of the client of a service. Onion routing, TOR's central mechanism, requires multi-hop forwarding and cryptographic operations at each relay server. TOR is therefore well-suited to web browsing but completely ill-suited for high-bandwidth video delivery. Fully decentralized, gossip-based broadcast protocols such as PAG [22] allow hiding

the source and destination of messages, but similarly come at a high cost, due to the use of resource-intensive homomorphic hashing.[1] This is also not compatible with VoD bandwidth requirements. Some video streaming services with privacy as a design goal have been proposed [18, 29, 40, 50]. All these solutions target fully *peer-to-peer* approaches, without core servers. This results in limited guarantees in terms of QoE: Video discovery is a best effort and unreliable operation, and the lack of a reliable authoritative source means videos of low popularity are only served with very low reliability.

Contributions. We present the design and implementation of PrivaTube, a *practical* and *privacy-preserving* VoD streaming system.

Practicality refers to the ability of PrivaTube to serve video content with a high QoE to its users: Low startup times, a constant and stable stream of high-bitrate video, and no interruption in the playback. This performance is enforced by the use of an edge-assisted CDN, allowing clients to fetch video content from *both* core servers and several *assisting peers* having accessed the same video in the past.

PrivaTube extends MS-Stream [11], a protocol for video streaming using multiple sources, compatible with the leading MPEG-DASH standard [53]. It ensures that the load on the core servers is minimized, and that the impact of faults is masked through redundancy.

PrivaTube preserves the privacy of its users by enforcing δ-unlinkability between specific users and videos, for a chosen value of δ. Access histories are masked from the untrusted infrastructure hosting core servers and from other clients by leveraging Intel SGX Trusted Execution Environments (TEEs) at both the client and server sides. PrivaTube further prevents assisting peers from inferring histories based on assistance requests by introducing *fake* requests. Fake requests have a cost, that is turned to the system profit by using them for pre-fetching content onto assisting peers and improving availability. This positively impacts QoE, in particular for low-popularity videos, and improves PrivaTube scalability.

We implement PrivaTube and deploy it on a distributed testbed with up to 14 SGX-enabled servers and clients to evaluate its performance and behavior. We also perform large-scale simulations based on a real-world data set of video access histories. Our results show that PrivaTube leverages multiple sources and fake requests to improve QoE, and compares favorably to non-privacy-preserving streaming.

The paper is organized as follows. We present an overview of the constituents and privacy objectives of PrivaTube in Section 2. We detail how the system scales and provides high QoE in Section 3, and how it preserves privacy in Section 4. We discuss our contributions and provide a security analysis in Section 5. Our evaluation is presented in Section 6. We review related work in Section 7 and conclude in Section 8.

2 System model and objectives

We start by detailing the service model and system constraints, followed by the adversarial model and privacy objectives, that guide the design of PrivaTube.

[1] To disseminate a 1,080p video, PAG would require each node to perform 7,200 costly homomorphic hashes per second. PAG further consumes three times the bandwidth of the transferred payload for control messages necessary to enforce accountability.

Service model. We target the Video on Demand (VoD) service model, consisting of a video player in a web browser allowing users to select and play videos from a publicly-known catalog. The objective is to reach the highest possible Quality of Experience (QoE). Achieving high-QoE delivery is a multi-criteria optimization. It consists in (1) provisioning a content bitrate that is not only the *highest* possible but also the *stablest* possible, (2) minimizing the amplitude and occurrence of variations in quality, (3) avoiding video interruptions and (4) ensuring a fast startup time. Continuity and stability of the video playback, together with fast startup times, are the factors that impact the most the viewing experience of users [52].

Deployment constraints. We target VoD providers who do not wish to monetize the personal data of their users while requiring good scalability and reasonable operational costs, e.g. open and alternative social media such as PeerTube [46]. The provider uses public cloud infrastructures to host servers for metadata and video content. For cost reasons, it does not use a third-party CDN. The number and capacity of cloud servers are limited. In particular, upload bandwidth for servers is not sufficient to successfully provision all clients with high-quality video at a reasonable cost.

Security assumptions. We assume that users trust their own machine but do not trust the other machines on which PrivaTube runs, i.e., the public cloud infrastructure and the other users. However, we assume that each node participating in PrivaTube is equipped with an Intel SGX-enabled processor. We believe that this is a reasonable assumption given the increasing availability of such processors on commodity hardware and cloud offerings (e.g. Microsoft Azure). We assume that the code running inside SGX enclaves is trusted (e.g., it does not contain bugs, backdoors). The trust in enclave code can be the result of its certification by a trusted third party, e.g. the open-source community. We assume that all used cryptographic primitives are trusted and that the adversary does not have enough computational power to forge them.

Privacy objective. Our privacy objective is to prevent an adversary from being able to exploit video access histories of any user in the system. This requires concealing the actual access history, i.e. legitimate events related to the actual visualization of a video by a user should not be collectible by the adversary in clear. The objective of PrivaTube is to achieve a good privacy-utility tradeoff. It must limit the exposure of personal data to the adversary on the one hand, and maintain cost-effectiveness and practicality (respect of high QoE demand), on the other hand.

Adversary model. We assume an adversary that aims at breaking the privacy guarantee offered by the system, i.e., uncovering the interest of users for specific video items. To reach this objective, we assume the strongest possible adversary (in terms of means) that is a *global* and *active* adversary. Global means that the adversary can monitor and record the traffic on all network links. Active means that the adversary can control all infrastructure nodes in the cloud, and run up to f client nodes to reach its objective. However, we assume that the adversary does not aim at breaking the system operation (e.g., by running denial of service attacks).

Our system design is detailed in Section 3, while privacy preservation is the focus of Section 4.

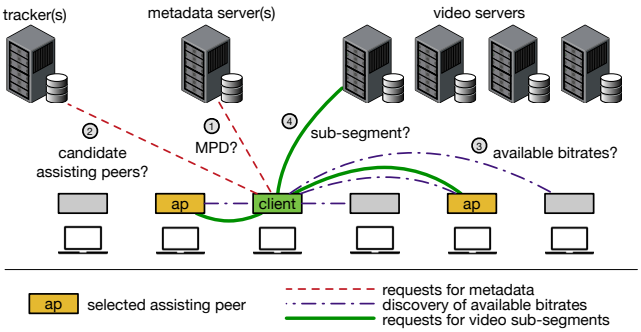


Figure 1. Streaming using video servers and assisting peers.

3 Practical and High-QoE Streaming

We detail the architecture of PrivaTube and how *adaptive* and *multi-source* streaming enables it to reach a high QoE.

3.1 Edge-assisted CDN

To address the limited capacity of dedicated servers in providing video content to users, we leverage an edge-assisted CDN. Figure 1 illustrates the architecture of PrivaTube, without privacy enforcement. Video content is obtained from a combination of *video servers* and *assisting peers*. Client nodes keep a cache of previously accessed videos, and may be selected to act as assisting peers by other clients.

Video servers in PrivaTube are stable but come in limited numbers. Assisting peers have limited bandwidth, and may leave the system at any time. Enforcing high QoE under these constraints leads to the following requirements. First, single servers or assisting peers may not be able to provide alone the highest quality to a client. This requires the ability to stream simultaneously from multiple sources. Second, faults and disconnections may result in video interruptions. This requires some redundancy in the obtained video content, enabling to switch back to lower-bitrate content rather than stopping the video. Finally, the quality of network connections may fluctuate during a video playback session. This requires a streaming protocol that seamlessly *adapts* to network conditions, and that we describe next.

3.2 Adaptive Streaming

MPEG-DASH (*Dynamic Adaptive Streaming over HTTP*) is the leading standard for delivering video. It uses HTTP on top of TCP or QUIC. DASH is deployed by companies such as Netflix, YouTube or Twitch. Video content is split into segments of a few seconds (usually one to ten seconds) for which different qualities (i.e. different bitrates) are available. Clients obtain the list of versions and servers hosting them from a metadata server returning a manifest file, or MPD, and download segments directly from video servers.

A DASH client can dynamically switch between bitrates to adapt to changing network conditions, e.g. when its download capacity decreases. The client maintains a buffer of video segments. The adaptation uses a combination of the buffer size and the prediction of future download times, with the objective of avoiding *rebuffering*.

Multiple-source streaming. DASH is efficient for serving videos from well-provisioned servers in the cloud. However, for edge-assisted video delivery from peers with less stable or reliable links, fetching from multiple sources with some level of redundancy

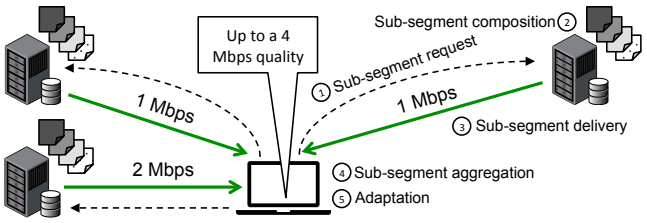


Figure 2. Multi-source adaptive streaming with MS-Stream.

is a strong asset. The PrivaTube streaming protocol extends a multiple-source adaptive streaming protocol, MS-Stream [11], fully compatible with DASH. MS-Stream is illustrated by Figure 2, while Figure 1 presents the complete workflow of our extension.

A MS-Stream client collects video content from multiple sources. It reconstructs segments of the highest-possible bitrate supported by its download link, even when none of the sources is able to individually provide this quality. It ensures availability through redundancy of video content with low-bitrate versions, that can be used as a backup and help avoid rebufferings.

For each segment, a MS-Stream client assembles individual *sub-segments* requests, using as many servers from the MPD as necessary to satisfy its target bitrate. Segments and sub-segments are formed of a sequence of short (e.g. 0.5 s) Groups of Pictures (GoPs). Each video server can serve a GoP in different bitrates, in both a low-quality (LQ) and *several* high-quality (HQ) versions. A sub-segment assembles GoPs, some in LQ and others in the HQ level requested by the client. This allows obtaining a HQ version of each GoP from exactly one server, while also redundantly requesting this same GoP in LQ from other servers. In the example of Figure 2, if the segment is formed of 4 GoPs, the client could ask the bottom-left server, with a capacity of 2 Mbps, for GoPs 1 and 2 in HQ and GoPs 3 and 4 in LQ. It would then request from the top-left server, with a capacity of 1 Mbps, GoP 3 in HQ, and GoPs 1, 2 and 4 in LQ. Similarly, the top-right server with the same capacity would return GoP 4 in HQ, and GoPs 1, 2 and 3 in LQ. The client assembles a segment with the highest received bitrate for each GoP. It uses the redundant GoPs in LQ as fallbacks should segments in HQ be missing. Bandwidth overheads depend on the bitrates used. We observe a 7.8% average increase of bandwidth usage with the parameters used in our evaluation.

Selection of assisting peers and servers. The selection of servers and sub-segments in the previous version of MS-Stream [11] exclusively favors QoE for the client, but does not consider different classes of servers. In PrivaTube, we wish to limit the use of video servers and favor the use of assisting peers. We extend MS-Stream for this purpose, as follows.

First, the use of assisting peers requires an additional service, the *tracker*. This server[2] keeps track of the video access history of clients. It returns to the client a random subset of up to 50 Candidate Assisting Peer (CAP) (② in Figure 1). For scalability reasons, the tracker does not maintain the association between CAPs, individual segments, and specific bitrates. Peers may indeed only have different qualities available for each segment, as a result

[2]We consider a single server for the tracker and for the metadata server in our implementation. The horizontal scaling of both servers can leverage stateless tiers and replicated NoSQL databases [32]. We keep this extension for future work.

of the adaptation policy. Registering this fine-grained information with the tracker would greatly impair scalability. Instead, clients register their access to the video with the tracker only *once*, and clients must *discover* for each segment what bitrates are available from the CAPs. This is done for each new segment (③ in Figure 1).

Second, we modify the selection of sources, and the associated selection of sub-segments, to favor the use of assisting peers over video servers. Following the quality discovery phase, the list of CAPs is pruned of peers who cannot offer the required HQ quality for the segment. The selection uses a greedy algorithm iterating over remaining CAPs and video servers. The selection considers first CAPs for which an estimation of the upload capacity is available locally. This corresponds to peers which were used for assistance in the past, and enables some stability in assistance relationships. Following this, the selection considers other CAPs, i.e., for which this estimation is not available. Video servers are finally considered if absolutely necessary. For each considered source, the selection determines the maximal number of GoPs that can be served by the peer in the target level of HQ, together with the other GoPs in LQ. This depends on the bandwidth capacity estimation for this source. For CAPs for which the information is unknown, a limited number (up to 4 over 12 in our implementation) of GoPs in HQ can be requested. For each selected peer, a random set of uncovered GoPs in HD is assigned in the corresponding sub-segment request, and the GoP is marked as covered with HD. The selection stops when all GoPs are marked, and the sub-segment requests are sent out (④ in Figure 1).

Improvements. The establishment of downloads from assisting peers has a higher latency than the direct download from video servers. In order to meet the QoE objective of fast video startup, the first segment is downloaded directly using the standard DASH procedure from a single video server.

We note that the effectiveness of selecting assisting peers instead of video servers depends on the video popularity, directly resulting in more copies at client peers. Unpopular content is at risk of being unavailable in the required quality in enough CAPs. We actually address this problem together with privacy preservation, as described in the next section. The concealing of users' interests indeed relies among other measures on the issuance of *fake* requests for content. We leverage these to the system's interest, implementing a cache *pre-fetching* strategy, and provisioning enough copies of all videos on client peers. This reduces the load on video servers, even for less popular content.

3.3 Implementation

The base system, without privacy protection, is implemented as follows. The client is written in JavaScript and runs inside a web browser. The metadata server is a key-value store. It hosts and delivers MPD manifests to the clients. The video servers are implemented in Java and store unencrypted video content. The tracker is implemented as an in-memory key-value store. All services are accessible through REST interfaces over HTTP, including communication between clients.

4 Privacy in PrivaTube

The goal of PrivaTube is to protect the access history of users (e.g. identified with their IP address) to videos. This history should not be exploitable *in the clear* by anyone else than the client node itself.

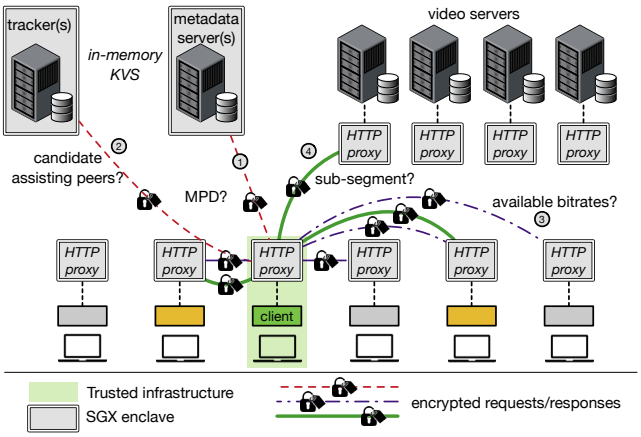


Figure 3. PrivaTube architecture for privacy preservation using HTTP proxies and servers inside SGX enclaves.

In order to better understand who can learn this information in PrivaTube, let us consider a user u interested in a video v. To this end, and following (a simplified version of) the steps described in Figure 1, u formulates a request *req* and sends it to the metadata server and to the tracker to collect the IP addresses of a set of video servers and candidate assisting peers from which it will get segments for reconstructing v. From these steps, and if no security mechanisms are used, one can easily see that a number of nodes in the system can learn the link between u and v. These include the tracker and the metadata server, using the information contained in *req*, and the video servers and assisting peers as they serve segments of v directly to u. Additionally, an adversary that listens to the network would also learn the link between u and v either from the content of *req* or from the segments of v that u downloads. Finally, an adversary that takes control of either of the above nodes would similarly learn the link between u and v.

In the following, we present the two main security principles that PrivaTube uses to protect the link between u and v. First, PrivaTube leverages Trusted Execution Environment (TEE) at *both* the client and server sides to prevent data leakage. Second, PrivaTube leverages *fake requests* to protect users access histories from insider attacks.

4.1 Trusted execution environments

TEEs offer guarantees of isolation, confidentiality, and integrity of data and computations performed in untrusted machines by leveraging custom microprocessor zones. We use Intel Software Guard Extensions (SGX) [16], which defines the concept of *enclave* as an isolated unit of data and code execution that cannot be accessed even by privileged code (e.g., the operating system or hypervisor). Enclaves can be attested, that is, it is possible to prove that the code running in the enclave is the one intended, and that it is running on a genuine Intel SGX platform. Once attested, enclaves can be provisioned with secret data by using authenticated secure channels. Moreover, enclaves can persist secret data outside the trusted zone by using a sealing mechanism.

Protecting the tracker. The tracker in PrivaTube stores information about nodes that have a local copy of a video. This information is clearly critical and any adversary taking control of the tracker

would immediately break the privacy property we aim at preserving. In order to prevent any information leakage, we run the tracker code inside a TEE. The tracker data is kept encrypted in a local key-value store, and can only be accessed in the clear by the code running inside the TEE enclave. Each tracker request only accesses a small subset of keys, henceforth the limited EPC memory size available to the enclave is not a limitation. The resulting SGX-enabled tracker guarantees that the access history of users to videos is protected even if an attacker takes control of the machine, of its operating system, or of the local hypervisor.

Protecting the metadata server. The metadata server stores for each video a manifest (MPD) that contains a description of the video and the list of servers from which its segments can be downloaded. A user u wishing to access a video v needs first to retrieve the manifest of v from the metadata server. To prevent any obvious linking between u and v, we run the metadata server code and we store its data inside a TEE, as for the tracker.

Protecting network traffic. As discussed above, an adversary listening to all network exchanges *in the clear* will learn the link between u and v. In order to make the message exchanges unobservable, all entities running in PrivaTube are put behind HTTP proxies running inside SGX enclaves. HTTP proxies intercept, encrypt and decrypt all inbound and outbound traffic as illustrated in Figure 3. Communicating proxies share a common encryption key which is securely transmitted to both communicating enclaves if and only if its remote attestation process succeeds. As such, an adversary listening to the network cannot learn the link between u and v as messages circulating between the various HTTP proxies are encrypted.

Using SGX-enabled HTTP proxies also allows protecting from an adversary taking control of the various entities participating in PrivaTube. Indeed, while a tracker or a metadata server can see an incoming request from u they can not have access to the content of this request nor to the content of the response sent back to u. Video servers and assisting peers serve video segments to u following requests forwarded by the proxy. We assume that video servers are serving many requests simultaneously, hence disallowing an adversary from precisely determining which specific incoming request at the proxy corresponds to which outgoing request for a video segment, an assumption done in similar systems such as Koi [28].[3] This is, however, not a property we can guarantee for assisting peers who process a limited number of video segment requests. We explain next how we mitigate this case using fake requests.

4.2 Fake requests

Using the above security mechanisms, u is able to stream v in a privacy-preserving manner. However, the link between u and v can still be revealed if an attacker runs its own client machine and starts issuing requests for v to uncover the community of users interested in that specific movie. This attacker could send requests for v to the tracker and collect the IP addresses of candidate assisting peers that possess segments of v (including u), thus uncovering the link between u and v. In order to mitigate this risk a key mechanism used in PrivaTube is the generation of fake requests. Specifically, each client participating in PrivaTube sends a given proportion of fake requests along with legitimate requests.

δ-unlinkability. Fake requests allow PrivaTube to guarantee δ-unlinkability between users and videos they are interested in. Enforcing δ-unlinkability means that at any point in time, the probability of guessing that a user u is interested in a video v is at most equal to $\delta \in [0, 1]$. Hence, the lower the value of δ the better the privacy of users. To enforce δ-unlinkability, a client c must maintain at any point in time t a number $\text{Fr}_t(c)$ of fake requests that is defined as $\text{Fr}_t(c) = \text{Lr}_t(c) * \frac{1-\delta}{\delta}$, where $\text{Lr}_t(c)$ is the number of legitimate requests it has issued up to t. For instance, let us assume that the system designer wants to enforce δ-unlinkability with δ equal to 0.5. Semantically, this means that the insider attack that learns a link between u and v can only infer that u is interested in v with a probability of 0.5, which is equivalent to flipping a coin. To enforce this property u would need to maintain $\text{Fr}_t(u) = \text{Lr}_t(u)$ at any point in time, which corresponds to sending as many fake requests as legitimate requests.

Generating fake requests. The difficulty when generating fake requests is to make them indistinguishable from legitimate requests. Towards this purpose, fake requests in PrivaTube are generated following the distribution of video popularities in the system. Relying on video popularity allows avoiding awkward/detectable behaviors such as a very unpopular video being requested too often (i.e., through fake requests). This behavior is not desirable because on the one hand, the request may be spotted as being a fake request by an adversary and on the other hand, replicating unpopular videos brings nothing useful to the system operation, creating way more copies of video segments than what is actually needed to ensure they will be available on assisting peers.

We implement two fake request generation policies, *pop* and *samePop*. All policies are executed in the tracker, running inside an SGX enclave. The *pop* policy generates fake requests by following the overall distribution of video popularity in the system. In this policy, the tracker keeps track of the number of requests issued for each video so far, which reflects their popularity. Every time a client issues a legitimate request, the tracker suggests a fake request following the distribution of video popularity, i.e., popular videos have a higher probability of being picked than unpopular ones. The *samePop* policy generates fake requests by following the *local* popularity of requested videos. That is, every time a user issues a request for a given video, the tracker suggests a video with a popularity similar to the requested one.

5 Discussion

We present a security analysis of PrivaTube with a focus on its privacy guarantees. Following this, we review compromises used in its design, and discuss limitations and possible mitigations.

5.1 Security Analysis

This section presents the security analysis of PrivaTube. We focus on the enforcement of the δ-unlinkability property. To this end, we consider whether the various entities participating in the system are able to break the property or not.

On the client side. The code running on the client side is divided into two parts, the HTTP proxy running inside an SGX enclave and

[3] We note that if this assumption does not hold, i.e. if there is insufficient concurrency for these requests, it is possible to modify the proxy to arbitrarily reorder the existing traffic together with *chaff* (fake) traffic. We leave, however, the implementation of such a feature to future work.

the streaming client running outside of the enclave. A malicious adversary running a client with the purpose of breaking other users' privacy cannot bypass the HTTP proxy and run man-in-the-middle attacks. However, it may issue specific requests using the PrivaTube protocol and then learn from which assisting peers it is downloading video segments. It can learn this by observing local traffic. However, thanks to the use of fake requests, the adversary will not be able to infer whether the assisting peers from which the video segments have been downloaded are effectively interested or not by the corresponding video.

Furthermore, up to f malicious clients under control of the attacker and aiming at weakening the δ-unlinkability property could modify the code of their local application to avoid sending fake requests in the system. By doing this, the overall number of fake requests in the system TFR_t at a given point in time t will be equal to:

$$\text{TFR}_t = (\text{TLR}_t \times \frac{1-\delta}{\delta}) - \sum_{i=1}^{f} \text{FR}_t(i)$$

where TLR_t is the overall number of legitimate requests sent in the system up to time t. However, while this decrease may have an impact on the replication factor of movies in the system, it will have no impact on the δ-unlinkability property of correct nodes. Indeed, as the proportion of fake requests in the local history of a correct node is preserved with respect to legitimate requests, an attacker that would uncover the link between this user and a given movie would not be able to distinguish with a confidence greater than δ whether the user is effectively interested in the video or not.

On the tracker side. The tracker code exclusively runs inside an SGX enclave. An adversary taking control of the tracker cannot bypass the SGX enclave. Furthermore, as the traffic incoming and outgoing from the enclave is encrypted, the only information that can be collected by an adversary is that a given node is issuing a request. If there is no other traffic when the user sends the request, the attacker may see from which video servers and assisting peers the client is gathering video segments. Using this knowledge, the attacker may try to learn the videos stored on these nodes by requesting them (in a brute force manner). However, this knowledge would be insufficient to guess which exact video was downloaded by the corresponding user and whether the latter was a legitimate or a fake request.

On the metadata server side. The reasoning is the same as for the tracker side. The metadata server holds important information about which video server holds which video segments. However, the processing of the request is performed inside an SGX enclave and the attacker may only learn the co-existence of the request and flows to video servers and assisting peers, and not determine which precise video was requested, and if it was a legitimate or fake request.

On video servers and assisting peers side. Video servers and assisting peers store video segments and serve these segments to requesting users. But similarly to the metadata server and the tracker server, requests on these nodes are handled inside SGX-enabled HTTP proxies. Hence, an attacker taking control of these nodes will not be able to bypass the enclave and learn what video segments are served to which specific user. A malicious video server or assisting peer could nevertheless delete its local videos keeping a single (or a set of) semantically sensitive video(s). As such it would learn the set of nodes requesting the latter. This threat is mitigated by the use of fake requests as the adversary will not be able to get a confidence greater than δ regarding the legitimate interest of the corresponding users on this video.

5.2 Limitations

We discuss in this section the limitations of our system and how we expect to mitigate them in our future work.

On the generation of fake requests. The generation of fake requests is an important mechanism in PrivaTube as it allows us to enforce δ-unlinkability for correct users. However, to be effective, fake requests must be forged in a way that makes them indistinguishable from legitimate requests. PrivaTube uses movie popularity for the generation of fake requests (i.e., popular movies have a higher probability to be selected as fake requests than unpopular ones). The probability distribution of legitimate requests targets will be similar to the probability distribution of fake ones. However, this policy does not capture particular access patterns to videos (e.g., users accessing a collection of movies in sequence such as the episodes of a given series). These sequential accesses could be used to probabilistically distinguish legitimate from fake requests. A solution would be to replace the generation of fake requests at the level of a single video by generation of *fake access logs*, copying the access behavior of another user. This may hinder the use of fake requests to implement proactive pre-caching of video segments using fake requests. We will consider this approach in our future work.

On the use of Intel SGX enclaves on the client side. Assuming the use of Intel SGX enclaves on the client side could be a limitation to the deployment of PrivaTube, in particular on portable devices. However, trusted execution environments are becoming common place even in handheld devices (e.g., ARM TrustZone [5]) which increases the likelihood for the adoption of PrivaTube in the near future.

On the presence of freeriding assisting peers. Freeriders are a well-known threat to collaborative systems. A freerider is a node that benefits from the system without contributing its fair share to it. In the context of PrivaTube, assisting peers could freeride by (for instance) turning off the application each time they do not watch video streams. If a large portion of assisting peers behave as such, there might be an impact on the overall QoE. We consider this problem as orthogonal to the one considered in this paper. Nevertheless, the problem of freeriders has been widely studied in the context of collaborative systems in general and in the context of live streaming in particular [6, 22, 26, 34, 59]. We plan on evaluating similar mechanisms (e.g., incentives or accountability mechanisms to track freeriders) in a future version of PrivaTube.

On the integrity of videos served by assisting peers and video servers. Assisting peers and video servers could misbehave by replacing their video content with junk videos. Mitigating this threat can be done by performing integrity checks on the client side (e.g., using `md5sum`).

On the provision of video recommendations to users. PrivaTube does not support the provision of video recommendations to its

users. The literature contains various research works in this direction, which could serve as a starting point for integrating such functionality while preserving privacy. These solutions rely either on adding differentially private noise [37] or on the use of cryptographic primitives [27].

On compromised client Intel SGX Enclaves. Our design relies on the proper implementation of Trusted Execution Environments (TEEs). The Intel SGX TEE that we use in our implementation has been shown to be vulnerable under certain conditions to side-channel attacks [33, 58]. The protection against such attacks is an orthogonal concern to the design of PrivaTube. We expect future iterations of Intel SGX to address these limitations, and other future implementations of the TEE concept to avoid them *by design*. For current SGX-enabled processors, solutions such as Varys [42] may be used to prevent side-channel attacks from being exploited. A complementary approach, that we leave for future work, is to limit the attack surface of potential leaks of enclave private keys, after an attack is observed or suspected (e.g., using a detection system such as Déjà Vu [13]). This requires periodically rotating the long-term secrets provisioned to PrivaTube client enclaves. Each rotation could spawn a Diffie-Hellman key agreement session, similarly to the Intel SGX attestation procedure. This mechanism would achieve *forward secrecy*: compromising a long term enclave private key does not compromise the session keys.

6 Evaluation

We proceed to the evaluation of PrivaTube. Our evaluation combines the analysis of the performance and behavior of a prototype deployed over up to 14 SGX-enabled servers and clients, and large-scale simulations based on a real-world data set of video access histories.

We wish to answer the following research questions:

- What is the impact of proxy-ing the content requests through Intel SGX TEEs' enclaves on throughput and latencies achievable by PrivaTube video servers? (§6.2)
- Is the use of assisting peers successful in improving QoE for the clients, and does it help to reduce the load on video servers? (§6.3)
- Are fake requests effective in hiding clients' access patterns to videos, and in improving performance and usefulness of assisting peers? (§6.4)

6.1 Experimental setup

The PrivaTube prototype is deployed on a cluster of 14 SGX-enabled Intel NUC nodes. Each node is an 8-core Intel i7 processor at 3.50 GHz with 32 GB of RAM. Nodes are connected in a LAN using 1 Gpbs Ethernet. The setup supports the use of network emulation to emulate WAN links using the `tc` tool. Network emulation parameters for video clients follow the DASH test case #2a (NP2a) [20]. This profile is based on observations of the typical characteristics of network clients at DASH providers: an incoming bandwidth of about 4 Mbps, a latency to the video servers of about 100 ms, and a packet loss of about 7%. The network emulation for servers caps their uplink bandwidth to 10 Mbps.

We use a video of 1 minute and 42 seconds, with 17 segments of 6 seconds. The video is available on video servers in three qualities: LD, SD, and HD. The low definition LD (bitrate of 512 Kbps) is used as the backup LQ quality in PrivaTube. LD segments are about 250 KB in size *after encoding*.[4] The SD and HD are the two available high-quality (HQ) variants. As detailed in Section 3, clients may choose the HQ variant that matches their download capacity. The SD (standard) definition (bitrate of 1 Mbps) leads to segments of about 500 KB after encoding. The HD (high) definition (bitrate of 4.1 Mbps) corresponds to the *full HD* quality. Segments in HD are about 2 MB after encoding.

Clients maintain a buffer of 30 seconds, i.e. upon starting a playback they download 5 segments (the first one from video servers, the following ones from a combination of video servers and assisting peers) and fetch a new segment as soon as one buffer slot is consumed.

We compare PrivaTube to two baselines. The first baseline is the standard DASH protocol. Our DASH implementation uses the NGINX high-performance HTTP server [41]. The second baseline is PrivaTube without enabling any of the mechanisms for protecting users' privacy. This corresponds to the system as described in Section 3, without any of the additions detailed in Section 4. We call this version **ClearTube**, to emphasize that exchanges happen *in the clear*. In more detail, ClearTube does not use HTTP proxies running inside SGX TEEs, does not encrypt any of the exchanges, and does not use fake requests.

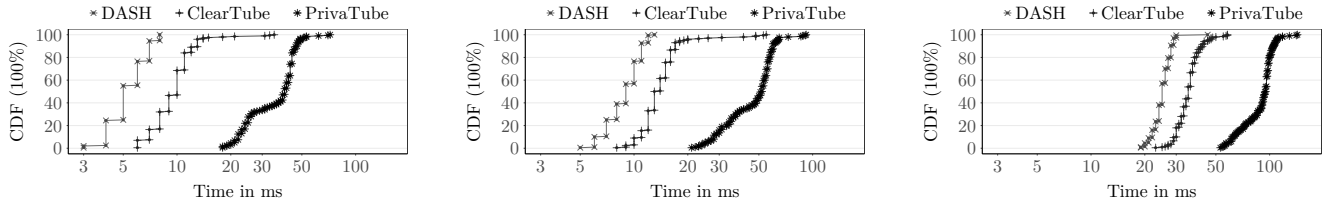
6.2 Performance of video servers

We start by evaluating the impact of privacy preservation on the performance achievable by a *single* video server. This performance is the primary measure of cost-effectiveness in any DASH deployment. For a VoD provider, the achievable bandwidth with one video server directly impacts return-on-investment (RoI). More specifically, we wish to assess if the use of request proxying through SGX does not impair too much the performance of each video server.

We do not use network emulation in this experiment. The comparative performance of PrivaTube and ClearTube enables to isolate the overhead of using SGX-based HTTP proxies. The difference between ClearTube and DASH allows us to isolate the impact of requesting sub-segments and the need to assemble them at the video server level (whereas DASH is able to directly serve pre-assembled segments).[5]

We first focus on request *latencies*. We set up a single client performing 200 consecutive requests for a 6-second video segment. Figure 4 presents the cumulative distributions of retrieval delays for the three available bitrates, using the three systems. Unsurprisingly, DASH provides the lowest latencies, and latencies increase with the segment size. The overhead of assembling sub-requests at the video server is highlighted by the performance of ClearTube. For LD segments (250 KB), the DASH median efficiency (20 ms to serve 1 MB) is twice that of ClearTube (40 ms to serve 1 MB). For larger segments however, this difference is attenuated, e.g. for HD segments (2 MB), the median efficiencies are 12.5 ms *vs.* 17.5 ms to serve 1 MB. The difference between ClearTube and PrivaTube is more important, and is a result of using SGX for the HTTP proxy. The establishment of a link between the enclaves at the client and the server, the exchange of secrets for establishing the secure channel, and the encryption of communications, all have an impact on latency. For LD segments, the median efficiency of PrivaTube (160 ms to serve 1 MB) is 25% of the median efficiency

[4] The exact size depends on the codec and nature of the video.
[5] One could cache pre-computed segments at a PrivaTube video server, but we did not implement this optimization.



(a) LD segments **(b)** SD segments **(c)** HD segments

Figure 4. Cumulative distribution of latencies over 200 requests for a 6-second segment in various bitrates in DASH, ClearTube and PrivaTube without using network emulation. Note that the abscissa uses a logarithmic scale.

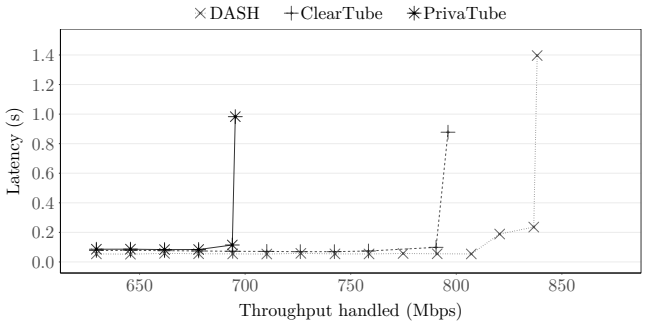
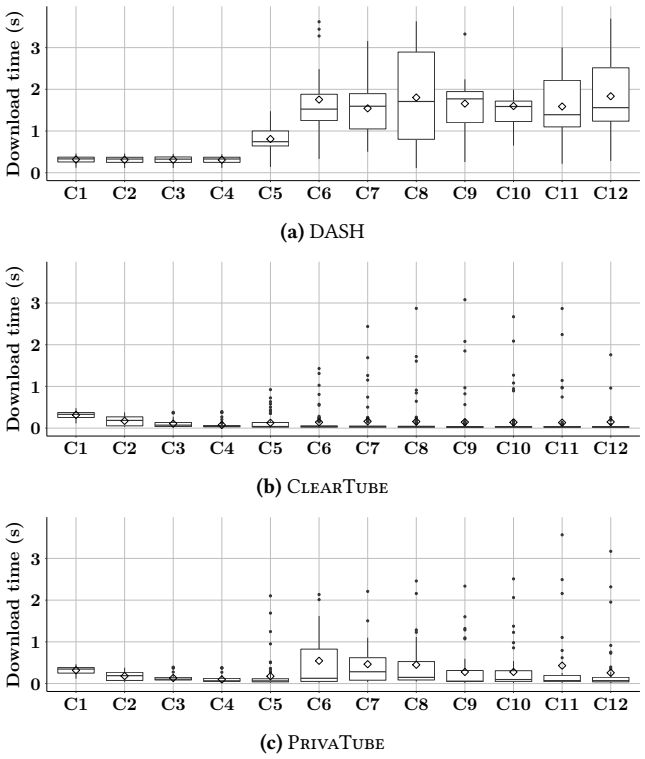


Figure 5. Throughput and latency for DASH, ClearTube and PrivaTube, without using network emulation. The inflection shows the saturation point of the three solutions.



(a) DASH

(b) ClearTube

(c) PrivaTube

Figure 6. Distributions of segments download times

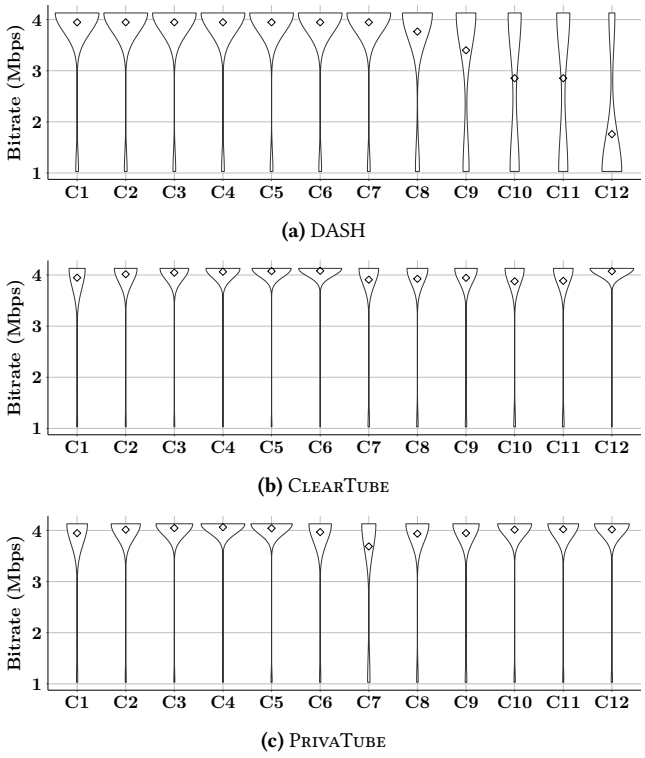
of ClearTube (40 ms). However, this difference is significantly reduced for larger bitrates. For HD segments, PrivaTube reaches an efficiency difference of 36% (47.5 *vs.* 17.5 ms to serve 1 MB). Despite the unavoidable overheads linked with a higher level of security, PrivaTube achieves median latencies of 95 ms (average 89 ms) for the HD bitrate, which remains negligible in practice for 6-second segments.

We now focus on achievable throughput for a video server under a concurrent request workload. We use only the HD quality, with video segments of size 2 MB. We set up a client with the `wrk2` workload injection tool for HTTP requests [57]. We use 4 injection threads from a single client, after checking that this setup is enough to saturate all three configurations. Figure 5 presents the achieved latencies for the three solutions under an increasing number of requests. The latency in the non-saturated case (e.g., with a total throughput of 650 Mbps) is close to the latencies for single requests (Figure 4c). We observe an inflection point for the three systems, indicating that the server is no longer able to serve incoming requests on time and reaches its maximal capacity. Again, the difference between DASH and ClearTube allows isolating the costs of the extra protocol steps in PrivaTube, without the use of the SGX HTTP proxy. ClearTube reaches saturation at 796 Mbps, 95% of the max capacity of DASH (838 Mbps). Comparing PrivaTube with ClearTube allows isolating the cost of the SGX HTTP proxies and of end-to-end encryption. PrivaTube saturates at 695 Mbps, 87% of ClearTube, and 83% of DASH. The saturation point of a single PrivaTube server corresponds to 260 clients consuming full-HD content, while ensuring privacy and enabling the use of assisting peers, against 312 clients with DASH. Again, the added value of privacy protection justifies this overhead.

6.3 Impact of assisting peers

In this second experiment, we evaluate the complete PrivaTube infrastructure. We use the 14 nodes. We use one node to host the tracker, and one node for the metadata server. To emphasize the limited capacity of the VoD provider infrastructure, we use a single video server. The remaining 12 nodes are used as clients, denoted as C1, C2, . . . , C12. We use the network emulation settings described in our evaluation setup. As before, we compare DASH, ClearTube and PrivaTube.

Initially, the video is only available at the video server in all three qualities (LD, SD and HD). Clients initiate streaming sessions to obtain and play the entire video. Each session starts by the download of the first segment from the video server, followed by



(a) DASH

(b) ClearTube

(c) PrivaTube

Figure 7. Distribution of achieved playback bitrates

the sequential request for 4 more segments from the video server and assisting peers, if any are available to provide the content. This allows filling the 5 initial slots in the client's buffer.

Client C1 starts at time 0, followed by C2, C3 and C4 each with a 10-second interval. 10 seconds after the start of C4 (40 seconds after C1), we emulate a *flash crowd* where the remaining 8 clients are started in sequence without additional inter-arrival delays. We observe two key performance indicators. The download time for individual segments indicate if the system operates in a non-saturated mode, and whether clients are subject to resource contention. The achieved quality rate is the effective bitrate at which clients were able to play the video. It is directly linked with the download capability: The adaptation automatically switches to a lower bitrate when the target bitrate cannot be obtained without a risk of rebuffering. There were no rebuffering in our experiments, but the three protocols differ greatly in achieved quality rate and its stability, directly impacting QoE.

Figure 6 presents the distribution of segment download times, while Figure 7 presents the achieved playback rates. We observe only very minor variations in achieved metric values over different runs and therefore report values over a single, randomly-selected one.

We observe that DASH performs well for the first 4 clients. Indeed, these clients are free to download and fill their buffers from the video server without interference with the other peers: The server has an upload of 10 Mbps, hence requiring 8 seconds to fill the 8 MBs of the buffer with video in HD. However, when the number of client increases we observe a tremendous increase in download times for individual segments, indicating that the server is not able to catch up with the requested throughput. The direct effect is that some clients (C9 to C12) adapt their requested bitrates to avoid a rebuffering. The QoE for these clients decreases significantly.

The general performance of ClearTube and PrivaTube follow the same trend. In concordance with our previous experiments, we observe additional latencies for segment download times with PrivaTube compared to ClearTube, due to the use of SGX-supported proxies (Figures 6b and 6c). When client C2 starts its session, C1 has already downloaded the first six segments from the video server and is selected as assisting peer by C2 for segments 2 to 5 to the extent of its upload bandwidth capability (capped to 4 Mbps). When peer C3 starts, both C1 and C2 are available, and so on. We observe in Figure 6b that the median download time actually decreases with more peers, and therefore more potential sources, but also results in more outliers for clients that start at the same time. The largest outlier is for the first segment, that has to be retrieved from the video server under contention, and due to competing requests towards the same assisting peers. The scenario of the addition of 200% more peers is a worst-case one; yet, PrivaTube succeeds in keeping the average and median download times to less than a second. The result on QoE is immediately visible on the achieved playback rates (Figure 7). All clients achieve a very stable (narrow) distribution with high bitrates. Differences between ClearTube and PrivaTube are negligible in terms of QoE, and the experiment demonstrates the impact of using multiple sources and assisting peers on maintaining playback quality even during a sudden increase in the number of clients.

6.4 Fake requests and pre-fetching policies

We finally evaluate the beneficial impact of fake requests on QoE. We discussed the privacy impact of fake requests in Section 5.1. We focus here on how fake requests' ability to pre-fetch content onto clients enables improving availability and the general utility of assisting peers. Our evaluation is based on simulations, in order to be able to use a large dataset denoting the interest of users in movies.

Dataset. Access histories to large VoD services are not public for obvious privacy reasons, and it would be unethical to exploit the lack of privacy of existing services to collect such data. We build instead video access histories from publicly-accessible data. More specifically, we use the complete year of 2014 of the open and non-commercial MovieLens [30] movie rating network.[6] MovieLens allows cinema enthusiasts to rate movies and enable personalized recommendations. 7,763 users produced 39,177 ratings for 4,283 distinct movies in 2014.

The cumulative distribution of movies popularity in MovieLens is presented in Figure 8. It is a heavy-tail distribution typical of VoD systems [14, 63]. 50.42% of the movies were rated only once, while the most popular was rated 64 times. Obviously, this represents only a sample of actual accesses and interests in movies, as only a small fraction of users rate movies they watch on MovieLens. We posit however that this fraction is a uniform sample, making this dataset statistically representative of what a sampling of actual accesses to videos in a large-scale VoD system would yield.

We build a *timeline* of accesses to videos from the MovieLens dataset. We consider the rating of a given movie by a user as a

[6] We chose 2014 as it is the last available full year from MovieLens 20M dataset (with ratings from January 9, 1995 to March 31, 2015).

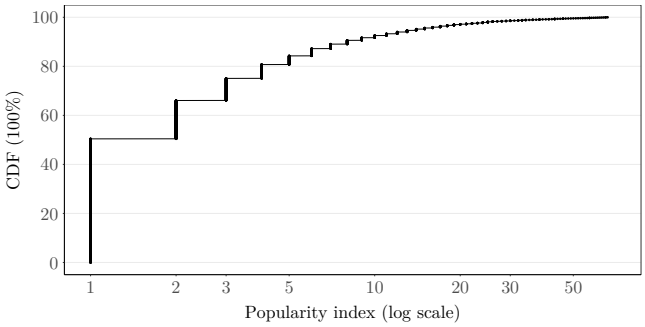


Figure 8. Distribution of movies popularities in MovieLens.

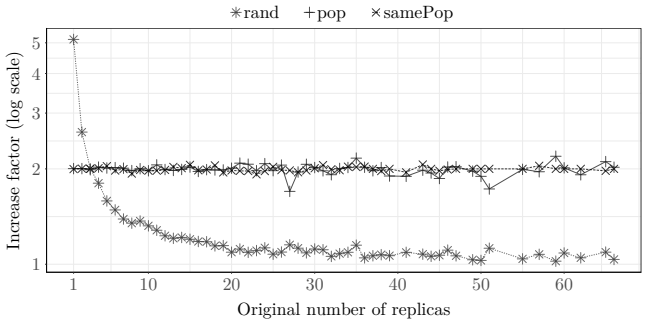


Figure 9. Replicas increase factor, $\delta = 50\%$

timed event, denoting the access to the video. The series of events is used as an input to the simulator.

Simulation. We emulate both legitimate requests (i.e. access events from the MovieLens data set) and fake requests. We implement the two policies for generating fake requests detailed in Section 4 in the tracker: *pop* considers the distribution of popularities of previously requested videos, and draws a random one according to this distribution; and *samePop* specifically picks a random video with the same level of popularity as the one requested in the legitimate request. In addition, we consider a third *baseline* policy, in order to highlight the impact of considering past access histories when generating fake requests: *rand* simply selects a random video from the list of all previously requested videos, regardless of their popularity.

We are interested in the increase in availability that fake requests allow. Ideally, we would like the number of copies of each video to be sufficient to allow downloading most segments from assisting peers rather than from the video servers.

We split the data set of accesses in two periods, with accesses done from January to the end of September 2014 in the first period, and accesses made in the remaining 3 months in the second period. We use the first period to compute popularities in the system, as the number of requests for each video. The popularity for a video v at the end of the first period is denoted as $v_{1^{st}}$. We then replay the accesses in the second period, and for each access, use the selected fake requests policy. At the end of the second period, we record the number of copies of the video as $v_{2^{nd}}$. We define the *replica increase factor* as the ratio $v_{2^{nd}}/v_{1^{st}}$. Figure 9 presents the distribution of this metric, as a function of $v_{1^{st}}$, when exactly one fake request is sent for each video access ($\delta = 50\%$).

Table 1. Replicas increase factor for various values of δ

# fake req.	$\delta = 66\%$ × 0.5			$\delta = 50\%$ × 1			$\delta = 25\%$ × 3		
	Mean	Median	Std. Dev.	Mean	Median	Std. Dev.	Mean	Median	Std. Dev.
rand	2.3	1.7	1.4	3.5	2.5	2.6	8.6	6.0	6.9
pop	1.5	1.5	0.5	2.0	2.0	0.6	4.0	4.0	1.1
samepop	1.5	1.5	0.5	2.0	2.0	0.6	4.0	4.0	1.1

We can observe that the *rand* policy results in a heavy bias towards increasing the availability of low-popularity videos. This is not surprising, as these videos dominate in the data set, and are therefore more likely to be selected by a random draw. On the other hand, *rand* only marginally increases the number of replicas for the rest of the distribution. The *pop* and *samePop* policies, on the other hand, are effective at increasing the number of replicas regardless of the original popularity of the video.

We present aggregate results for two other values of δ, the probability to link a user to a video, in Table 1. We can observe that the target number of pre-provisioned copies of each video is achieved with great stability for both *pop* and *samePop*, but with a high skew for *rand*. There is no significant advantage in availability between *pop* and *samePop*. It is therefore sensible to favor *samePop*, to also hide the individual distribution of access video popularities for each individual user. The *samePop* policy is also more stable. It deviates less than *pop*, according to the results.

7 Related work

We review work on streaming platforms architectures, and on approaches to preserve privacy in streaming systems.

Streaming platforms architectures. We distinguish two families of streaming platforms, those targeting *live* streaming (i.e. the broadcast of a single stream, as it is generated) and those targeting the VoD model, as PrivaTube.

Live streaming does not require maintaining a cache for a collection of videos, making it adapted to fully decentralized streaming [35]. Exchange of video segments may happen between peers over an unstructured network using gossip-style interactions, as in CoolStreaming [61], or use dissemination structures built over an overlay network, as in MDC [44] or SplitStream [12]. Peer-to-peer live streaming is highly scalable, but it is difficult in practice to guarantee high QoE due to uncertainties in peer availability and network stability.

Supporting the VoD model requires being able to quickly discover a specific video in a catalog, and ensure that copies of each video exist in the system at all times, making it less amenable to a fully decentralized implementation. Instead, several authors proposed to complement a limited set of servers with edge peer resources as we do in PrivaTube. Push-to-Peer [55] targets a deployment on controlled networks, including for instance set-top-boxes deployed by an ISP. Push-to-Peer proactively pushes content to edge nodes to increase availability and QoE, based on a utility

model [56]. This is similar to the use of fake requests in PrivaTube but Push-to-Peer does not consider privacy aspects. P2VoD [25] combines source servers with the use of multicast trees [12, 44]. Video content is cached at clients who can join the trees and act as providers. BASS [19] is a similar approach using the BitTorrent protocol. Both P2VoD and BASS are only evaluated using simulations, and the ability of the peer-to-peer network to provide high QoE remains limited.

PrivaTube adopts an edge-assisted architecture where the discovery of video sources, including those at the edge, benefits from centralization. This pragmatic approach is in the interest of QoE, and eases the compatibility with the MPEG-DASH industry standard. A similar approach is used in Xunlei Kankan [64] and LiveSky [62]. Both systems combine CDNs with edge resources and report serving tens of millions of users simultaneously.

PeerTube [46] is an open social network for video sharing. PeerTube is decentralized and uses a collection of support sites, leveraging W3C's federation protocol ActivityPub [60], but the download of videos is made using a single server. The scalability, cost effectiveness and privacy guarantees of PrivaTube would make it an ideal protocol for the PeerTube social network.

Privacy-preserving streaming. Popcorn [29] uses private information retrieval (PIR) techniques to conceal the access to videos by clients of a VoD streaming platform. The platform only keeps encrypted video content, and PIR hides the actual interest in a video by forming requests that combine data from the entire catalog. However, Popcorn has a weaker adversarial model than ours as it requires non-colluding servers. Additionally, the overhead of the cryptographic mechanisms it uses makes it unpractical for large scale deployments (e.g., a distributed catalog of thousands of movies). Instead, the elastic architecture of PrivaTube, and the use of lightweight cryptographic mechanisms makes it scalable by design.

Rajan et al. [50] leverage a privacy-preserving publish and subscribe [43] communication system to hide the interests of clients from video providers, under the live streaming model. As with the use of onion networks (e.g. in TOR [24]) this approach requires cryptographic operations at intermediary nodes, which incurs latencies and costs likely to degrade QoE significantly.

Cui *et al.* [17] consider the more general problem of hiding access patterns to objects in a third-party CDN, and present encryption mechanisms under the Searchable Encryption (SE) model. This allows users to search and request items without revealing their interests in the clear. These mechanisms do not protect, however, from an attacker who would analyze the traffic to and from the clients. In contrast, PrivaTube does not require a specific and costly encryption of video content on video servers.

Previous approaches have proposed to add noise to legitimate traffic in order to conceal the interests of users, as fake requests allow in PrivaTube.

Decouchant et al. [21] developed concurrently to our work a technique similar to our use of fake requests, but for pure peer-to-peer video streaming, i.e. without any video servers. Peers subscribe to $k-1$ additional streams for each stream of interest, and participate to the dissemination of all k streams. Similarly to fake requests in PrivaTube, these additional participants increase the availability of videos. P3LS enables, however, peers to participate with less bandwidth to the dissemination of additional streams (up to 30% according to simulations), provided that an attacker is not able to determine with sufficient confidence a participation to the dissemination of a stream of interest from the dissemination of another stream. This property of *plausible deniability* [7] could also benefit PrivaTube.

Other approaches target other applications such as peer-to-peer file sharing. Swarmscreen [15] adds random connections in the BitTorrent file-sharing network to enforce plausible deniability. A noise level of 25% to 50% prevents an attacker from successfully mapping communities of interests in this network. However, Swarmscreen fake connections do not prevent an attacker from actively probing individual nodes for content and are not leveraged to improve content availability.

Mistrustful P2P [18] leverages erasure codes to reduce the cost of fake requests in a peer-to-peer file-sharing system and, as for PrivaTube, improve content availability while concealing users' interests.

Designing privacy-preserving systems using Intel SGX. Trusted Execution Environments (TEEs) and in particular Intel SGX have been extensively used in the past few years to build secure and privacy-preserving systems. Examples include replication services [10], Web search proxies [39, 47] or database systems [48, 51]. This demonstrates that TEEs are a promising technology as they offer a satisfactory compromise between security properties and performance overheads. While PrivaTube principles are not exclusively relying on Intel SGX properties, it still illustrates the benefits of this technology for performance-sensitive VoD applications.

8 Conclusion

We presented PrivaTube, a *practical* and *privacy-preserving* VoD streaming system. PrivaTube aggregates video content from multiple servers and edge peers to offer a high QoE for its users. It enables privacy preservation at all levels of the content distribution process. It leverages TEEs at servers and clients, and obfuscates access patterns using fake requests that reduce the risk of personal information leaks. Fake requests are further leveraged to implement proactive provisioning and improve QoE, filling two needs with one deed. We implemented PrivaTube and showed in an extensive evaluation of our prototype involving 14 SGX-enabled servers and clients that PrivaTube reduces the load on servers and increases QoE while providing strong privacy guarantees. Our future work includes the investigation of more sophisticated fake requests taking into consideration movie access patterns over time and the design of a private recommender system on top of PrivaTube.

Acknowledgments

We thank our shepherd, Mema Roussopoulos, and the anonymous reviewers for their comments. This work was partially funded by the French-German ANR-DFG project PRIMaTE (ANR-17-CE25-0017), and by the Belgian FNRS project DAPOCA (33694591).

References

[1] 2018. *Global Internet Phenomena Report.* Technical Report. Sandvine.
[2] 2019. *Cisco Visual Networking Index: Forecast and Trends, 2017–2022.* Technical Report. CISCO.
[3] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. 2012. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *Proceedings of IEEE INFOCOM.*

[4] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. 2012. Vivisecting YouTube: An active measurement study. In *Proceedings of IEEE INFOCOM*.

[5] ARM. [n.d.]. TrustZone: system-wide hardware isolation for trusted software. https://developer.arm.com/ip-products/security-ip/trustzone.

[6] Mira Belenkiy, Melissa Chase, C Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. 2007. Making P2P accountable without losing privacy. In *ACM workshop on Privacy in electronic society (WPES)*. ACM.

[7] Vincent Bindschaedler, Reza Shokri, and Carl A Gunter. 2017. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment* 10, 5 (2017), 481–492.

[8] Alessio Botta, Aniello Avallone, Mauro Garofalo, and Giorgio Ventre. 2018. A user-oriented performance comparison of video hosting services. *Computer Communications* 116 (2018).

[9] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the Netflix CDN. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018).

[10] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. Securekeeper: Confidential ZooKeeper using Intel SGX. In *17th ACM/IFIP/USENIX International Middleware Conference*.

[11] Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Negru, Jordi Mongay Batalla, and Eugen Borcoci. 2018. Adding a new dimension to HTTP Adaptive Streaming through multiple-source capabilities. *IEEE MultiMedia* 25, 3 (2018).

[12] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. 2003. SplitStream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM.

[13] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjà Vu. In *2017 ACM Asia Conference on Computer and Communications Security*.

[14] Ann L Chervenak, David A Patterson, and Randy H Katz. 1995. Storage systems for movies-on-demand video servers. In *14th Symposium on Mass Storage Systems*. IEEE.

[15] David R Choffnes, Jordi Duch, Dean Malmgren, Roger Guierma, Fabian E Bustamante, and Luis Amaral. 2009. Swarmscreen: Privacy through plausible deniability in P2P systems. *Technical report, Northwestern EECS* (2009).

[16] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[17] Shujie Cui, Muhammad Rizwan Asghar, and Giovanni Russello. 2017. Privacy-preserving content delivery networks. In *42nd Conference on Local Computer Networks (LCN)*. IEEE.

[18] Pedro Moreira da Silva, Jaime Dias, and Manuel Ricardo. 2016. Mistrustful P2P: Privacy-preserving file sharing over untrustworthy Peer-to-Peer networks. In *IFIP Networking Conference*.

[19] Chris Dana, Danjue Li, David Harrison, and Chen-Nee Chuah. 2005. BASS: BitTorrent assisted streaming system for video-on-demand. In *7th Workshop on Multimedia Signal Processing*. IEEE.

[20] DASH Industry Forum. [n.d.]. Guidelines for Implementation: DASH-AVC/264 Test cases and Vectors. https://dashif.org/docs/DASH-AVC-264-Test-Vectors-v1.0.pdf.

[21] Jérémie Decouchant, Antoine Boutet, Jiangshan Yu, and Paulo Verissimo. 2019. P3LS: Plausible Deniability for Practical Privacy-Preserving Live Streaming. In *38th International Symposium on Reliable Distributed Systems (SRDS)*.

[22] Jérémie Decouchant, Sonia Ben Mokhtar, Albin Petit, and Vivien Quéma. 2016. PAG: Private and accountable gossip. In *36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE.

[23] Jie Deng, Gareth Tyson, Felix Cuadrado, and Steve Uhlig. 2017. Internet scale user-generated live video streaming: The Twitch case. In *International Conference on Passive and Active Network Measurement (PAM)*. Springer.

[24] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[25] Tai T. Do, Kien A. Hua, and Mounir A. Tantaoui. 2004. P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *IEEE Intl. Conf. on Communications (ICC)*, Vol. 3.

[26] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. 2010. Lifting: lightweight freerider-tracking in gossip. In *11th ACM/IFIP/USENIX International Conference on Middleware*.

[27] Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, Mahammad Valiyev, and Jingjing Wang. 2017. I know nothing about you but here is what you might like. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[28] Saikat Guha, Mudit Jain, and Venkata N Padmanabhan. 2012. Koi: A location-privacy platform for smartphone apps. In *9th USENIX conference on Networked Systems Design and Implementation (NSDI)*.

[29] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. 2016. Scalable and private media consumption with Popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[30] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (2016), 19.

[31] Miltiadis Kandias, Lilian Mitrou, Vasilis Stavrou, and Dimitris Gritzalis. 2013. YouTube user and usage profiling: Stories of political horror and security success. In *International Conference on E-Business and Telecommunications*. Springer.

[32] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. ShieldStore: Shielded In-memory Key-value Storage with SGX. In *14th EuroSys Conference (EuroSys)*. ACM.

[33] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *26th USENIX Security Symposium*.

[34] Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. 2006. BAR gossip. In *7th symposium on Operating systems design and implementation (OSDI)*. USENIX Association.

[35] Yong Liu, Yang Guo, and Chao Liang. 2008. A survey on peer-to-peer video streaming systems. *Peer-to-peer Networking and Applications* 1, 1 (2008), 18–28.

[36] Dixin Luo, Hongteng Xu, Hongyuan Zha, Jun Du, Rong Xie, Xiaokang Yang, and Wenjun Zhang. 2014. You are what you watch and when you watch: Inferring household structures from IPTV viewing data. *IEEE Transactions on Broadcasting* 60, 1 (2014).

[37] Frank McSherry and Ilya Mironov. 2009. Differentially private recommender systems: Building privacy into the Netflix prize contenders. In *15th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[38] Ricky KP Mok, Vaibhav Bajpai, Amogh Dhamdhere, and KC Claffy. 2018. Revealing the Load-Balancing Behavior of YouTube Traffic on Interdomain Links. In *International Conference on Passive and Active Network Measurement (PAM)*. Springer, 228–240.

[39] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. 2017. X-search: Revisiting private web search using Intel SGX. In *18th ACM/IFIP/USENIX Middleware Conference*.

[40] Alok Nandan, Giovanni Pau, and Paola Salomoni. 2004. GhostShare: reliable and anonymous P2P video distribution. In *IEEE Global Telecommunications Conference Workshops*.

[41] NGINX Inc. [n.d.]. NGINX web server. https://www.nginx.com.

[42] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In *2018 USENIX Annual Technical Conference*.

[43] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. 2016. Confidentiality-preserving publish/subscribe: A survey. *ACM computing surveys* 49, 2 (2016).

[44] Venkata N. Padmanabhan, Helen J. Wang, and Philip A. Chou. 2003. Resilient peer-to-peer streaming. In *11th IEEE Intl. Conf. on Network Protocols (ICNP)*.

[45] Peer5. [n.d.]. Last mile delivery: Ensure coverage in underserved regions and internal networks with peer-assisted delivery. https://www.peer5.com/p2p.

[46] PeerTube. [n.d.]. A decentralized video hosting network, based on free/libre software. https://joinpeertube.org/en/.

[47] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. 2018. CYCLOSA: Decentralizing Private Web Search Through SGX-Based Browser Extensions. In *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE.

[48] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. Enclavedb: A secure database using SGX. In *Symposium on Security and Privacy (SP)*. IEEE.

[49] Quanteec. [n.d.]. Streaming with Quanteec: Quality and cost. https://quanteec.com.

[50] MA Rajan, Ashley Varghese, N Narendra, Meena Singh, VL Shivraj, Girish Chandra, and P Balamuralidhar. 2016. Security and privacy for real time video streaming using hierarchical inner product encryption based publish-subscribe architecture. In *Workshops of the 30th International Conference on Advanced Information Networking and Applications (WAINA)*. IEEE.

[51] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *Symposium on Security and Privacy*. IEEE.

[52] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 469–492.

[53] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (Oct. 2011), 62–67.

[54] Streamroot. [n.d.]. Powering the next generation of video delivery. https://streamroot.io.

[55] Kyoungwon Suh, Christophe Diot, Jim Kurose, Laurent Massoulie, Christoph Neumann, Don Towsley, and Matteo Varvello. 2007. Push-to-peer video-on-demand system: Design and evaluation. *IEEE Journal on Selected Areas in Communications* 25, 9 (2007).

[56] Bo Tan and Laurent Massoulié. 2013. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM Transactions on Networking (TON)* 21, 2 (2013).

[57] Gil Tene. [n.d.]. WRK2: HTTP benchmarking tool. https://github.com/giltene/wrk2.
[58] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium*.
[59] Xavier Vilaça, Luís Rodrigues, João Silva, Hugo Miranda, Gustavo Correia, and Tiago Maurício. 2018. FastRank: Practical lightweight tolerance to rational behavior in edge assisted streaming. *Pervasive and Mobile Computing* 46 (2018).
[60] W3C. [n.d.]. ActivityPub decentralized social networking protocol. https://www.w3.org/TR/activitypub/.
[61] Susu Xie, Bo Li, Gabriel Y Keung, and Xinyan Zhang. 2007. Coolstreaming: Design, theory, and practice. *IEEE Transactions on multimedia* 9, 8 (2007), 1661–1671.
[62] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. 2009. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *17th ACM international conference on Multimedia*.
[63] Hongliang Yu, Dongdong Zheng, Ben Y Zhao, and Weimin Zheng. 2006. Understanding user behavior in large-scale video-on-demand systems. In *ACM SIGOPS Operating Systems Review*, Vol. 40.
[64] Ge Zhang, Wei Liu, Xiaojun Hei, and Wenqing Cheng. 2015. Unreeling Xunlei Kankan: Understanding hybrid CDN-P2P video-on-demand streaming. *IEEE Transactions on Multimedia* 17, 2 (2015).