High-QoE Privacy-Preserving Video Streaming

Simon da Silva

université de BORDEAUX

LABORATOIRE BORDELAIS DE RECHERCHE EN INFORMATIQUE LaBRI

THESE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ DE BORDEAUX

École Doctorale Mathématiques et Informatique

Spécialité Informatique

Simon Da Silva

DIFFUSION VIDÉO AVEC UNE MEILLEURE QUALITÉ D'EXPÉRIENCE ET RESPECTANT LA VIE PRIVÉE

HIGH-QOE PRIVACY-PRESERVING VIDEO STREAMING

Sous la direction de :

Daniel Négru
Laurent Réveillère

Soutenue le mercredi 07 octobre 2020

Membres du jury :

Sonia Ben Mokhtar Directeur de Recherche, LIRIS-CNRS ..... Invitée
Sara Bouchenak Professeur, INSA Lyon ..... Rapporteuse
Yérom-David Bromberg Professeur, Université de Rennes ..... Examinateur
Pascal Desbarats Professeur, Université de Bordeaux ..... Président du jury
Daniel Négru Maître de Conférences, Université de Bordeaux ..... Co-directeur de thèse
Evangelos Pallis Professeur, Université Hellénique Méditerranéenne ..... Rapporteur
Laurent Réveillère Professeur, Université de Bordeaux ..... Co-directeur de thèse

Résumé

Titre : Diffusion Vidéo avec une Meilleure Qualité d'Expérience et Respectant la Vie Privée

La diffusion vidéo devrait atteindre 82% du trafic total sur Internet en 2022. Il y a deux raisons à ce succès : la multiplication des sources de contenu vidéo et la démocratisation des connexions haut débit à Internet. Les principales plateformes de streaming vidéo dépendent d'infrastructures planétaires pour répondre à la demande croissante en qualité visuelle. Cependant, l'utilisation de ces plateformes génère des données personnelles sensibles (sous la forme d'historiques de visionnage). Protéger les intérêts des utilisateurs est nécessaire pour une nouvelle génération de services de streaming vidéo respectueux de la vie privée.

Cette thèse propose une nouvelle approche pour du streaming vidéo temps-réel multi-sources en délivrant du contenu avec une meilleure qualité d'expérience (délai de démarrage rapide, flux stable en haute qualité, pas de coupures) tout en permettant une protection de la vie privée (grâce aux environnements d'exécution de confiance).

Mots-clés : Streaming, Sécurité, Vie Privée, Cloud, Informatique de confiance, Qualité d'expérience

---

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Unité Mixte de Recherche CNRS (UMR 5800)
351 cours de la Libération, 33400 Talence, France

Abstract

Title: High-QoE Privacy-Preserving Video Streaming

Video streaming is expected to exceed 82% of all Internet traffic in 2022. There are two reasons for this success: the multiplication of video sources and the pervasiveness of high quality Internet connections. Dominating video streaming platforms rely on large-scale infrastructures to cope with an increasing demand for high quality of experience and high-bitrate content. However, the usage of video streaming platforms generates sensitive personal data (the history of watched videos), which leads to major threats to privacy. Hiding the interests of users from servers and edge-assisting devices is necessary for a new generation of privacy-preserving streaming services.

This thesis aims at proposing a new approach for multiple-source live adaptive streaming by delivering video content with a high quality of experience to its users (low start-up delay, stable high-quality stream, no playback interruptions) while enabling privacy preservation (leveraging trusted execution environments).

Keywords: Streaming, Security, Privacy, Cloud, TEE, QoE

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Unité Mixte de Recherche CNRS (UMR 5800)
351 cours de la Libération, 33400 Talence, France

Remerciements

En m'engageant dans la recherche académique, j'espérais partager ouvertement des techniques, des connaissances, de l'expérience, et travailler avec des hommes et femmes du monde entier. De nombreuses communautés de chercheurs existent et forment des cercles (malheureusement bien souvent fermés) mêlant collaboration et amitié. J'ai eu la chance d'intégrer l'une d'elles et de travailler avec des gens sympathiques, compétents et ouverts. Les travaux de recherche exposés dans ce manuscrit auraient donc été impossible sans l'aide, la vision et l'implication d'un nombre important de personnes, et le soutien, la présence, le réconfort de beaucoup autres.

Pour commencer, je tiens à remercier mes deux directeurs de thèse, Daniel et Laurent. Vous avez su me montrer une voie passionnante et me faire progresser pas à pas, en m'encadrant avec justesse et sagesse. Vous avez levé les verrous techniques, administratifs et financiers que j'ai pu rencontrer sur mon chemin. Grâce à votre confiance, j'ai présenté nos travaux, voyagé, et collaboré avec des gens d'horizons différents. Merci à Mathias. Tu as rendu possibles les différentes contributions présentées dans cette thèse, que ce soit par ton code, ton aide technique, ton soutien ou ta bonne humeur. Tu as été le meilleur des canards,[1] et m'a permis de garder une motivation inébranlable tout au long de ces années en étant toujours présent, attentif, attentionné et bienveillant. Merci à Joachim. Tu m'as beaucoup appris et fait gagner un temps précieux par tes retours, ton aide, tes contributions, et tout le travail effectué, sur lequel se basent beaucoup de nos contributions présentées dans ce document. Merci à Stefan, David, Éric, Moubarak et Hamza. Vous avez activement contribué à la réalisation de plusieurs des solutions, toujours dans la joie et la bonne humeur. Merci à Sonia et Étienne. Vous êtes à l'origine de nombreuses idées et d'une très grande proportion du contenu des articles. Votre expérience, vision et implication ont permis à PrivaTube et `PProx` d'exister, et d'être présentés sous leur forme actuelle. Merci à Guillaume. Ce fut un plaisir de travailler

[1] `https://fr.wikipedia.org/wiki/M%C3%A9thode_du_canard_en_plastique`

avec toi, pour ta compétence, ta disponibilité et ton implication. `PProx` est le fruit d'une collaboration scientifique et technique regroupant activement toutes les personnes impliquées, et n'aurait jamais existé concrètement sans ton dévouement. En outre, je tiens aussi à remercier tous les membres du laboratoire et de l'équipe, le personnel administratif et financier, et tous les gens de l'université qui ont égayé et/ou facilité mon quotidien ces dernières années.

Sur un autre registre, je ne serais jamais arrivé au bout de ce doctorat sans le soutien moral quotidien des doctorants du laboratoire. Merci à Rohan, mon compagnon de soir et de tout jour. Tu as été présent, attentif, attentionné, compréhensif, et complice de beaucoup d'aventures mémorables. Tu as égayé de nombreux épisodes de ma vie, et permis de réaliser des rêves, que ce soit par le biais de l'AFoDIB ou autrement. Merci à Léo, notre fidèle compagnon de bureau.[2] Ton optimisme, ta bienveillance et bonne humeur étaient très précieuses. Merci à tous les copains de l'association et d'en dehors : Alex, Christelle, Pierre-Étienne, Tina, Paul, Jason, Rémi, Julien, Jonathan, Luis, Raph, Jérôme, Rémi, Henri, Mohammed, Théodore, Samah, et les nombreux autres que j'oublie. Vous êtes tous géniaux et méritez chacun une double-dose de compliments.

Je tiens à remercier tous mes autres amis,[3] à qui je dois beaucoup. Merci à Rémi,[4] Titouan, Clément, Imane, Sylvain, Pierre, et tous les autres qui feraient déborder ces remerciements.[5] Les aventures, divertissements, conseils et tranches de vie partagées avec vous m'ont permis de me construire et, éventuellement, d'en arriver à écrire ce manuscrit.

Enfin, merci à ma famille, toujours présente, toujours compréhensive, toujours positive. Merci à mes parents, Françoise, Antonio, et Laurent, Dominique. Vous m'avez apporté un soutien sans faille et un confort moral, psychologique et matériel, tout au long de ma vie. Grâce à vous, j'ai toujours été libre de faire des études sans me préoccuper du reste, confiant et serein, avec toujours un endroit ou revenir, que ce soit physiquement, moralement ou verbalement. Merci à Caroline, ma pacsenaire, comparse de vie ici et là, pour tout ce que tu fais pour nous. Tu m'as fait découvrir un pan de la vie que j'ignorais, et ton soutien quotidien m'est inestimable. Merci au reste de ma famille, cousins, cousines, oncles et tantes, avec qui j'ai grandi dans une atmosphère saine et protectrice.

[2]Quand tu y étais.

[3]Un grand nombre des personnes citées précédemment sont évidemment mes amis.

[4]Encore un !

[5]Oups, ils ont déjà débordé de deux pages. Tant pis, continuons donc la liste : Merci à Auriane, Antoine, Florian, Chloë, Théophile, Paul, Pierre, Nathan, Bastien, Jérémy, Sylvestre, Dorian...

Comme ceux mentionnés ici le savent, j'aime communiquer et vulgariser les sujets important à mes yeux, que ce soit avec des gribouillis sur un tableau, avec des slides, en vidéo, ou juste à l'oral, assis dans un bar ou entre deux portes. Mais, indépendamment du niveau de vulgarisation que je pourrais utiliser, il me sera à jamais impossible de communiquer mes passions, occupations et préoccupations à certaines personnes. Je dédie donc ce manuscrit aux membres de ma famille qui m'auraient probablement toujours soutenu aveuglément mais sont, physiquement ou mentalement, partis trop tôt.

— Simon Claude Victor Da Silva

Contents

List of Acronyms

AS	Autonomous System
CAP	Candidate Assisting Peer
CDN	Content Delivery Network
DASH	Dynamic Adaptive Streaming over HTTP
GoP	Group of Pictures
HAS	HTTP Adaptive Streaming
ISP	Internet Service Provider
LRS	Legacy Recommendation System
MPD	Media Presentation Description
OTT	Over-The-Top
P2P	Peer-to-Peer
PIR	Private Information Retrieval
RaaS	Recommendation-as-a-Service
QoE	Quality of Experience
SGX	Software Guard Extensions
TEE	Trusted Execution Environment
VoD	Video on Demand

List of Equations

List of Figures

List of Tables

Chapter 1

Introduction

> *I've learned that people will forget what you said, people will forget what you did, but people will never forget how you made them feel.*
>
> — Maya Angelou

Many new trends appeared with the democratization of the Internet. Video contents began to increase both in quantity and in quality, thanks to platforms such as YouTube [You20], Vimeo [Vim20] or Dailymotion [Dai20], all providing income through ads and sponsors. People look for specialized contents, which they can now consume on several devices (desktop and laptop computers, tablets, smartphones, smart TVs, etc.) whenever they want to. For these reasons, Video on Demand (VoD) services first emerged, followed by live streaming. Over-The-Top (OTT) delivery, thanks to websites, apps or set-top-boxes, is currently the most widespread way to deliver video content to consumers.

1.1 Motivation

Video streaming represented more than 60% of all Internet traffic [San19] and 65% of worldwide mobile downstream traffic in 2020 [San20]. According to Cisco [Cis18], video traffic is experiencing a tremendous growth and is expected to exceed 82% of the total Internet traffic by 2022, and live video will grow 15-fold to reach 17% of all video traffic by 2022. Two reasons account 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. Most of the time, although traffic

increase is rightfully forecast, core networks capacities are not upgraded due to the high cost of such an operation. Major issues consequently arise with respect to the Quality of Experience (QoE) of such services. Providing a high and fairly shared among users QoE is thus a rising issue as servers and network links become overloaded.

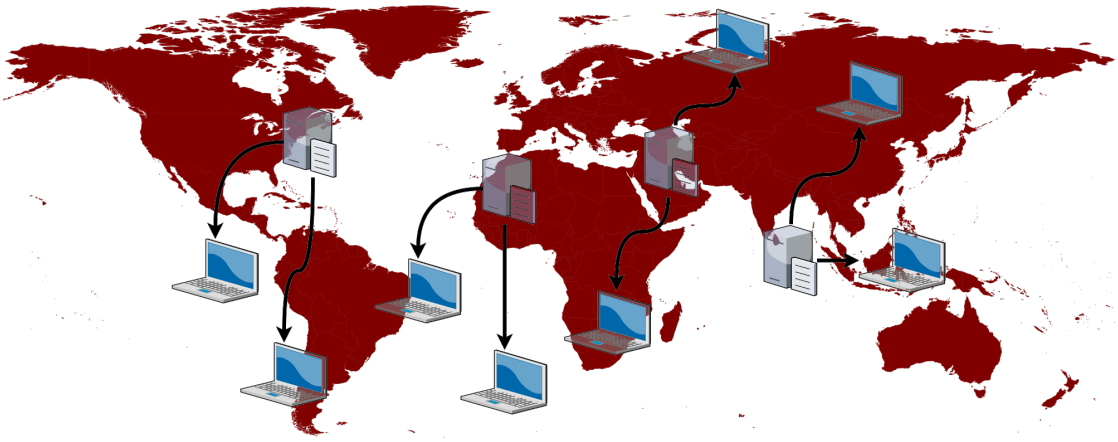


Figure 1.1: Content Delivery Network illustration

Dominating video streaming platforms rely on large-scale infrastructures to cope with an increasing demand for high QoE and high-bitrate content. **Content Delivery Networks (CDNs)** are extensively used for the delivery of video content over the Internet (see Figure 1.1). For instance, YouTube [You20], Netflix [Net20] or Twitch [Twi20] have set up planetary-scale proprietary CDNs [DTCU17, BCT+18, AJCZ12, AGH+12]. They further deploy extra CDN nodes directly inside Internet Service Providers (ISPs) networks (*e.g.*, Google Global Caches) and negotiate special peering relations with their Autonomous Systems (AS) [MBDC18]. Other platforms can rely on existing third-party CDNs to serve content. Dailymotion [Dai20] is reported to use the CDNs of Orange, Akamai and Limelight to scale video delivery in different parts of the world [BAGV18]. In such architectures, geographically distributed replica servers located as close as possible to the consuming clients are provisioned in advance with sufficient capacities using estimates of the expected workload. When accessing a content, consuming clients are automatically re-directed to the closest server to temper network congestion and achieve higher throughput.

In addition to CDN-based solutions, streaming services usually rely on **HTTP Adaptive Streaming (HAS)** solutions, such as the widely adopted Dynamic Adaptive Streaming over HTTP (DASH) standard from MPEG, or Apple's HTTP Live Streaming (HLS). These solutions enable consuming clients to dynamically adjust the requested

content bitrate according to the observed network conditions or to the client buffer occupancy (see Figure 1.2).

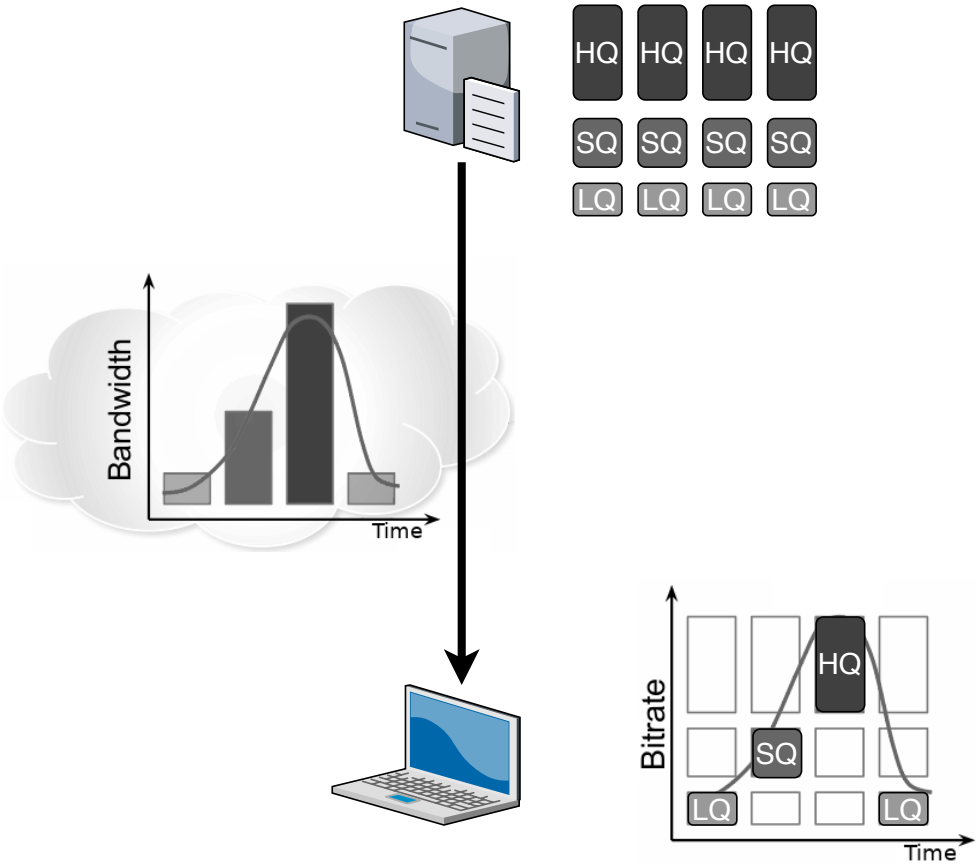


Figure 1.2: DASH illustration

However, as shown in Figure 1.3, if a large amount of end-users located under the same geographic area is simultaneously consuming the same streamed content, the nearest server can rapidly become overloaded. Some users may consequently suffer throughput degradation or content unavailability, and may experience a poor or unfairly shared QoE as they compete for limited network and server resources.

Although CDN solutions can handle a large volume of requests, they laboriously adapt to the highly dynamic and volatile nature of live streaming service audiences. As a consequence, the streaming infrastructure can rapidly be either over-scaled hence uselessly too expensive, or under-sized and thus delivering degradated QoE to end-users. Dedicated CDNs require a high up-front investment, and third-party CDNs incur high

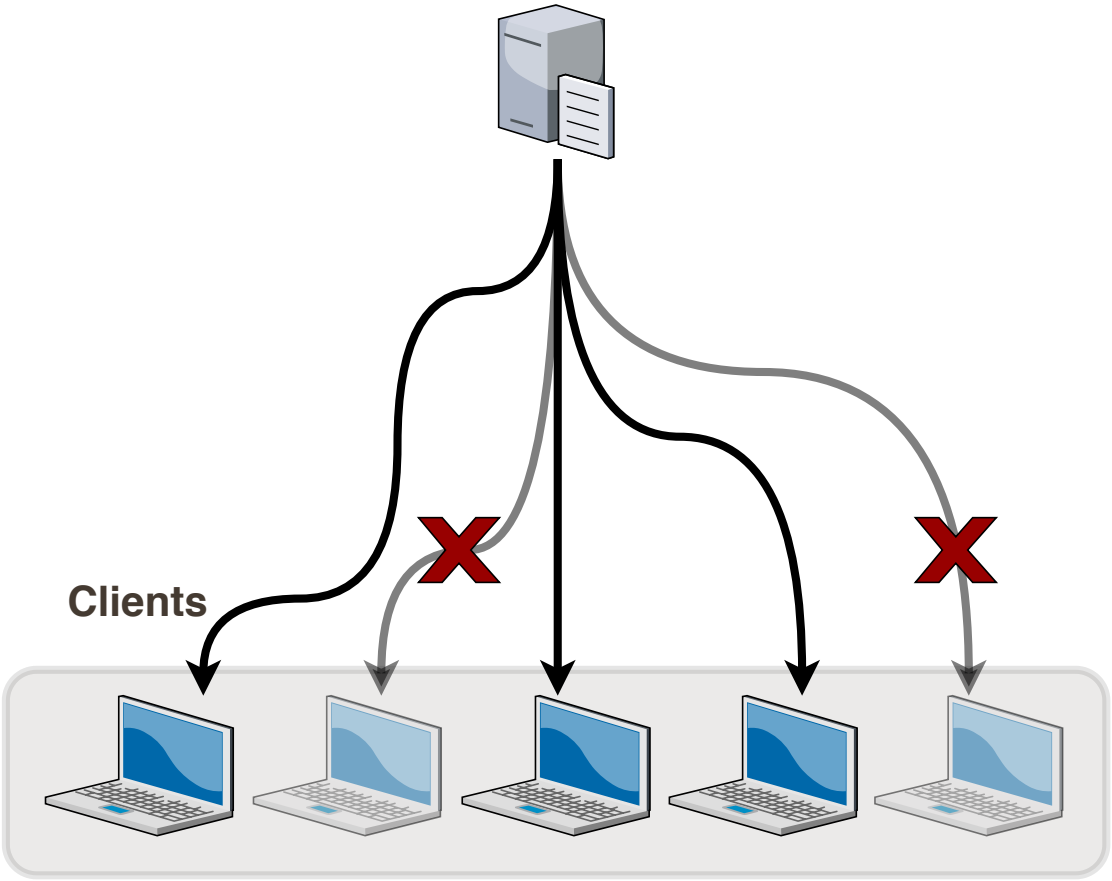


Figure 1.3: DASH congestion illustration

operational costs. The use of an **edge-assisted CDN** (see Figure 1.4) is an appealing alternative for smaller players or platforms that do not need to monetize their users' personal data to sustain their activity, such as the free and open PeerTube [Peeb] 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 [YLZ+09], Peer5 [Peea], PeerTube [Peeb], Quanteec [Qua], Streamroot [Str] and Kankan [ZLHC14].

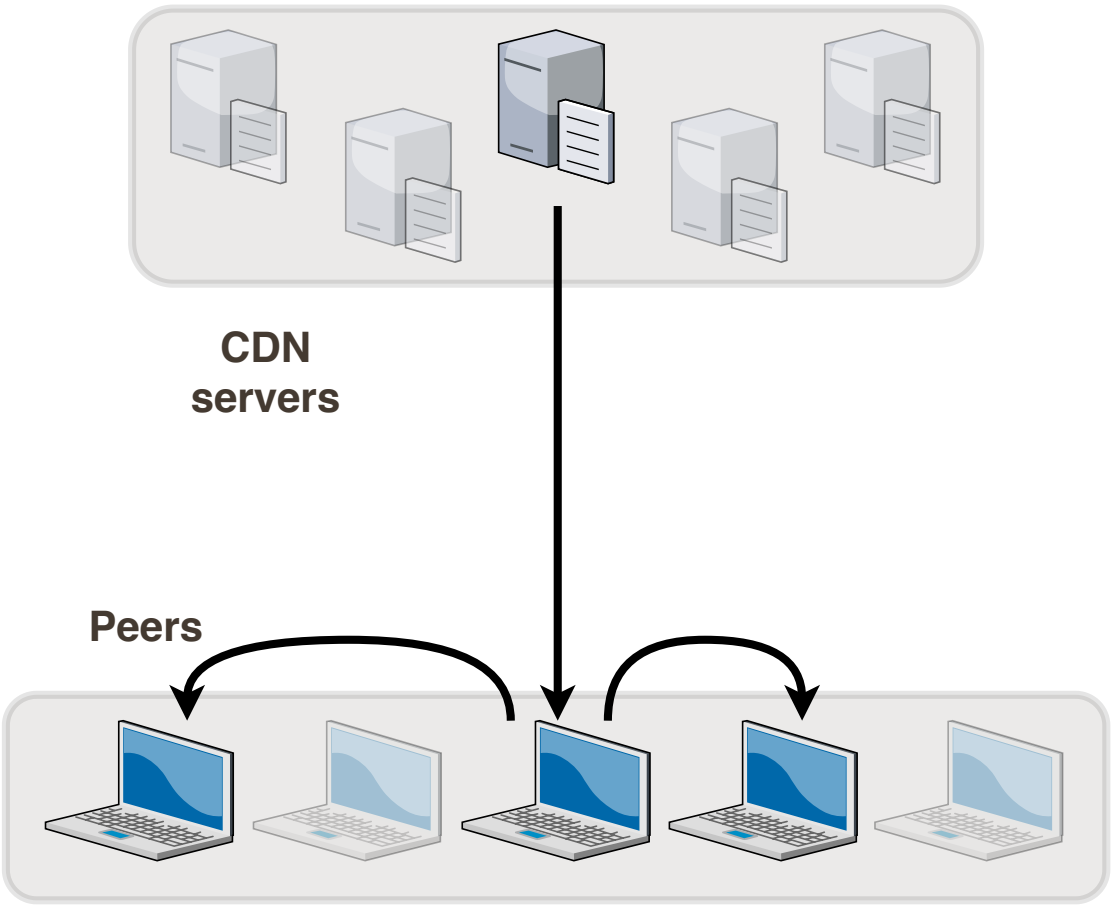


Figure 1.4: Edge-assisted Content Delivery Network illustration

Browsing on video streaming platforms generates an access history of watched content, specific for each user. This history 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 generation and availability of access histories also leads to major threats to **privacy** (see Figure 1.5).

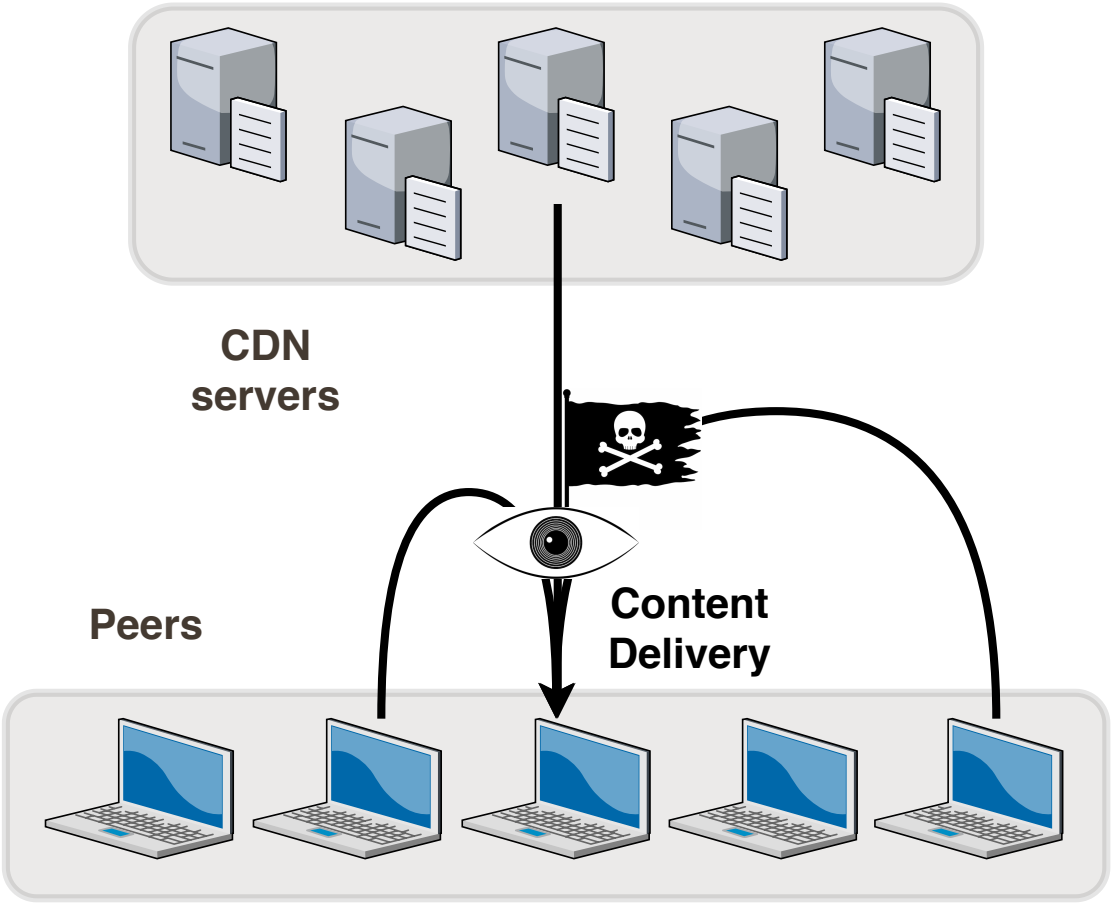


Figure 1.5: Privacy illustration

Indeed, this data can be used to infer private information about the user, such as his gender, his origin, and his political, religious or sexual orientation. Kandias *et al.* [KMSG13] show for instance that the political affiliation of YouTube users can easily be extracted from their access histories. Luo *et al.* [LXZ+14] similarly show how a household composition can be deduced.

1.2 Challenges and objective

Based on the current video streaming situation, two main challenges are foreseen to be tackled. The first one is to provide users with a high and fairly shared **QoE**. The second one is to protect users' **privacy** when using video streaming platforms.

Quality of Experience The target is an OTT video streaming service model, consisting of a video player in a web browser allowing users to select and play video content from a publicly-known catalog. 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 main factors that strongly impact the users' viewing experience [SES+14].

Privacy for video streaming Protecting users' privacy in a video streaming system requires hiding their access histories from providers and other users. Anonymizing networks such as Tor (*The Onion Routing*) [DMS04] allow to hide 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. Besides, fully decentralized, gossip-based broadcast protocols such as PAG [DMPQ16] allow to hide the source and destination of messages, but similarly come at a high cost, due to resource-intensive homomorphic hashing. PAG further consumes three times the bandwidth of the transferred payload for compulsory control messages to enforce accountability. This is also not compatible with video streaming bandwidth requirements. Some streaming services with privacy as a design goal have been proposed as well [GCM+16, RVN+16, dSDR16, NPS04]. All these solutions target fully Peer-to-Peer approaches, without core servers, which results in limited guarantees in terms of QoE, as video discovery is a best effort and unreliable operation, and the lack of a reliable authoritative source means that videos of low popularity are only served with very low reliability. The challenge is to provide a privacy-preserving video streaming solution which does not impair the Quality of Experience.

Objective This thesis aims at proposing a practical privacy-preserving video streaming system, providing both a high **Quality of Experience** and strong **privacy** guarantees to its users, without quality or performance degradation, at the lowest monetary cost.

1.3 Thesis overview

Our first approach towards increasing QoE and decreasing costs is `Muslin`, a dynamic server provisioning and advertising system. Then, we propose PrivaTube, the first video streaming system to provide strong privacy guarantees with unaltered QoE compared to traditional HAS streaming. Finally, we present `PProx`, a high-performance solution to provide privacy-preserving recommendations in video streaming systems.

The rest of this thesis is organized as follows:

- *Chapter 2 - Background* provides technical background about HTTP Adaptive Streaming and MS-Stream, a DASH standard extension to aggregate bandwidth from multiple sources to increase video quality and drastically reduce rebufferings.
- *Chapter 3 - Related work* presents related research work and the State of the Art of streaming platforms architectures, techniques for privacy-preserving streaming, and privacy-preserving recommender systems.
- *Chapter 4 - `Muslin`: High-QoE cost-efficient multi-source streaming* describes the `Muslin` solution. It uses dynamic server provisioning and advertising based on real-time delivery conditions combined with multiple-source video streaming to provide a better QoE at the lowest cost.
- *Chapter 5 -* PrivaTube*: Privacy-preserving edge-assisted streaming* details the design and implementation of PrivaTube. It offers a high QoE by aggregating video content from multiple servers and edge peers. Users' privacy is preserved through encryption in HTTP proxies running in Intel SGX enclaves, and fake requests to obfuscate access patterns. Fake requests are further leveraged to implement proactive provisioning and improve QoE.
- *Chapter 6 - `PProx`: High-QoE privacy-preserving Recommendation as a Service* presents `PProx` and how it complements video streaming systems. `PProx` provides pseudonymous and private recommendations with unaltered accuracy and excellent performance, as it supports arbitrary recommendation algorithms and has minimal deployment requirements. `PProx` design leverages an elastically scalable network of proxies to transparently pseudonymize users over a fleet of Intel SGX-enabled machines. `PProx` privacy guarantees are robust to even the corruption of one of the SGX enclaves.
- *Chapter 7 - Conclusion and further directions* concludes the thesis, summarizes the contributions and presents insight on possible further research directions.

Summary

Two main challenges are to be tackled: (i) to provide a high and fairly shared QoE to end-users; (ii) to protect users' privacy on streaming platforms.
This thesis aims at proposing a practical privacy-preserving video streaming system, providing both a high Quality of Experience and strong privacy guarantees to its users, without quality or performance degradation, at the lowest monetary cost.

Chapter 2

Background

You know, they say an elephant never forgets. But what they don't tell you is that you never forget an elephant.

— Bill Murray

A video is a sequence of pictures consecutively displayed to trick the human eye into seeing motion, which requires at least 16 frames per second. Nowadays, most movies display 24 to 60 frames per second, and up to several hundreds of images per second for some applications (*e.g.*, sports or video gaming). Consequently, as each picture can weigh up to a few megabytes, videos require a lot of data to be stored, delivered and displayed. This entails the use of compression algorithms, referred to as *codecs*.

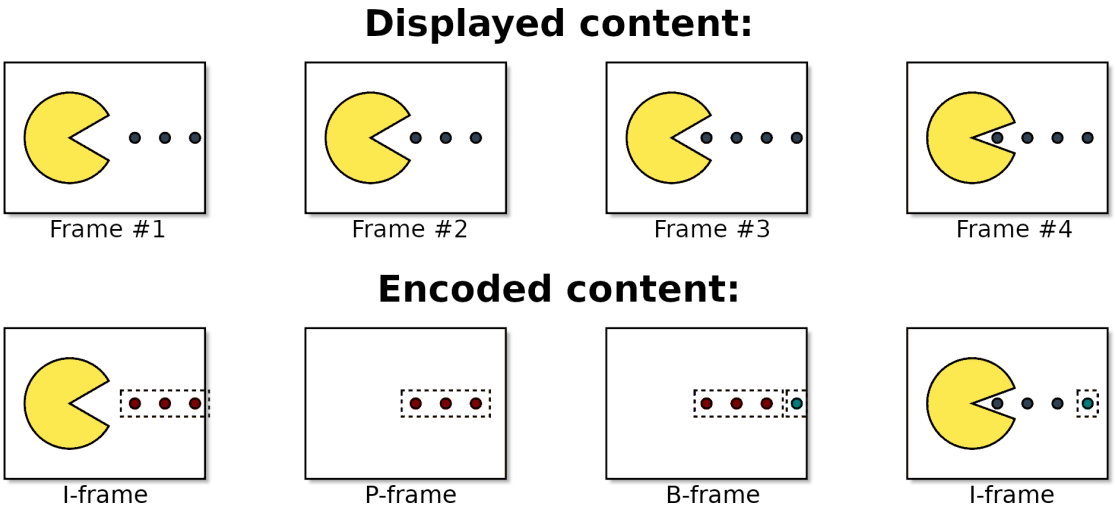


Figure 2.1: Video compression frame types

2.1 Traditional video encoding and delivery

Most codecs aim at reducing redundancy in videos (*i.e.*, wasted storage space) by computing movement vectors between consecutive frames and storing them instead of a whole picture. They traditionally use three types of frames: Intra (`I`) frames, which are standalone pictures; Predicted (`P`) and Bidirectional (`B`) frames, made of movement vectors computed from either previous (`P`) or both previous and next (`B`) frames (see Figure 2.1). The self-contained series of an `I`-frame followed by several `B` and `P` frames is called a Group of Pictures (GoP), and typically encodes a few hundred milliseconds of footage.

Video content delivery solutions have evolved a lot during the last three decades. Video streaming over the Internet appeared in the early 1990s. Many solutions were developed, such as RTP (Real-time Transport Protocol) / RTCP (RTP Control Protocol) [SCFJ03] in 1993, RTSP (Real Time Streaming Protocol) [SRL98] in 1993 and RTMP (Real-Time Messaging Protocol) [Ado20] in 1996. However, these solutions lack adaptation to changing network conditions, often resulting in playback stalls.

2.2 Adaptive Streaming

Since the late 2000s, HAS solutions have seen important interest in the industry and research, mainly due to their capabilities to render smooth video playback to the consumers, hence a better QoE. The overwhelming majority of OTT platforms now implement HAS solutions.

Various HAS solutions have emerged, such as Adobe HDS (HTTP Dynamic Streaming) [Ado18], Apple HLS (HTTP Live Streaming) [App18] or MSS (Microsoft Smooth Streaming) [Mic18]. In the early 2010s, the MPEG (Moving Picture Experts Group), formed by ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) and joined by most multimedia organizations, published the MPEG-DASH (Dynamic Adaptive Streaming over HTTP) standard [Sod11] in an attempt to federate all efforts towards a single solution.

2.2.1 Dynamic Adaptive Streaming over HTTP

The DASH standard, widely adopted in the industry and deployed by companies such as YouTube, Netflix, Facebook or Twitch, is now the leading standard for delivering

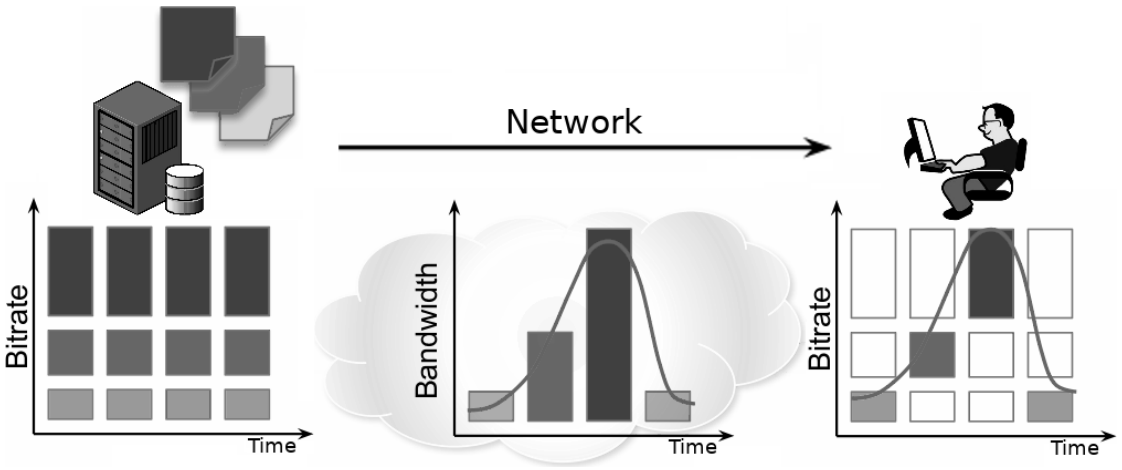


Figure 2.2: DASH standard content delivery overview

video (along with Apple's HLS, which is technically very similar). DASH aims at delivering uninterrupted multimedia content through the network via conventional HTTP (Hypertext Transfer Protocol) traffic [Sod11]. It uses HTTP on top of TCP (Transmission Control Protocol) or QUIC [Goo20].

As shown in Figure 2.2, video content is split into segments of a few seconds (usually one to ten seconds) containing several GoPs, similarly to traditional streaming. However, different qualities (*i.e.*, in terms of bitrates) are encoded and made available for each segment. A DASH client can then 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 rebufferings.

Media Presentation Description

A manifest file, the Media Presentation Description (MPD), details the representations that are available for every segment and also provides a list of servers where these segments can be accessed at. The MPD is initially handed out to the client, which then proceeds to retrieve the segments at the desired quality directly from video servers.

2.2.2 Other HAS solutions

Proprietary commercial systems such as HDS, HLS or MSS are following the same principles. The clients first download a manifest file, and then video segments. Technical

specifications do not enforce specific quality adaptation mechanisms. However, a few differences can be pointed out.

Manifests Every streaming session begins with the download of a manifest. In MSS, the manifest is an XML file. In HDS, the manifest is called the Adobe Media Manifest, or F4M. In HLS, manifests are text files called M3U8. The master M3U8 file contains global information about the media, and pointers to other M3U8 playlists. These playlists include the name and duration of every segment for each quality.

Video containers and codecs Some proprietary systems are limited by specific video containers. For example, segments in HDS should be in put into F4V containers. Until recently, HLS clients were consuming MPEG-TS segments. Nowadays, HLS can be used with fragmented MP4. Moreover, unlike DASH which is codec-agnostic and only limited by the external decoders available, proprietary systems may be codec-specific for both video and audio content. As an example, MSS, HDS and HLS are not compatible with the open-source codecs VP8 and VP9.

Proprietary implementations The official implementations from the technology owners may be the only solution to watch content if the standard is not compliant enough for open source or concurrent projects to emerge. In the early years, both HDS and MSS were restricted to the proprietary solution. Nowadays, HLS implementation in iOS cannot be modified and it is not possible for a developer to use his own implementation. Because of this limitation, proprietary HAS solutions may be constrained by the non-optimal adaptive mechanisms carried by the official implementations.

HAS limitations

The work of Adhikari et al. [AGH+12] advocates that QoE would greatly benefit from the venue of a practical HAS that can actually utilize multiple servers simultaneously. Even though there are some propositions for multiple servers streaming [ZLL15, PZC11], none of the existing approaches provide a high QoE through both redundancy between independent sub-segments (to avoid rebufferings) and bandwidth aggregation (to reach a higher visual quality).

2.2.3 Multiple-source streaming

DASH is very 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 is a strong asset.

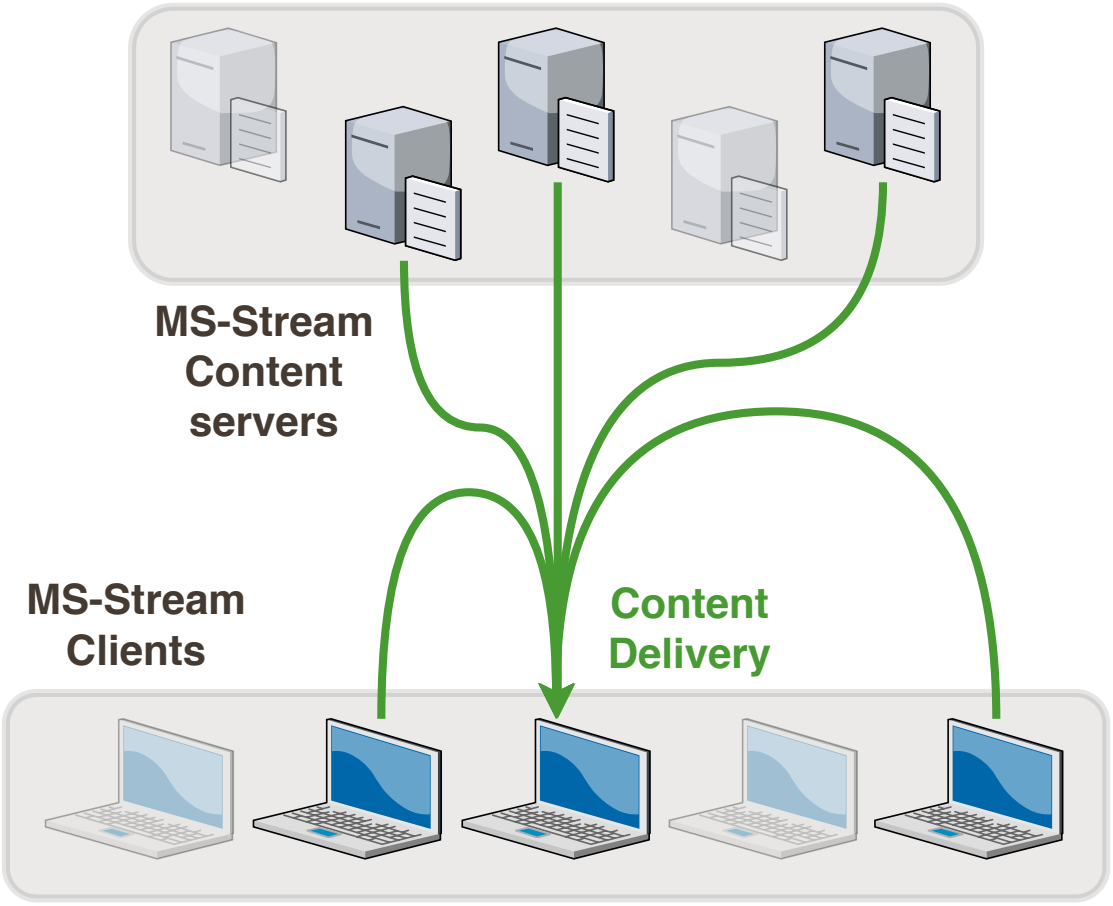


Figure 2.3: MS-Stream illustration

Multiple-Source Adaptive Streaming over HTTP

Multiple-Source Adaptive Streaming over HTTP (MS-Stream) [BQLN17a, BQLN+18, BQLN+17b, BQLN+17c] is a proposition that extends the DASH standard, wherein a client can simultaneously utilize multiple servers in order to aggregate bandwidth over multiple links while being resilient to network and server impairments (see Figure 2.3). 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 helps avoiding rebuffering events.

Sub-segments generation and composition

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 short (*e.g.*, 0.5 s) video frames sequences gathered into independent units called 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 to obtain a HQ version of each GoP from exactly one server, while also requesting this same GoP in LQ from other servers as fallback.

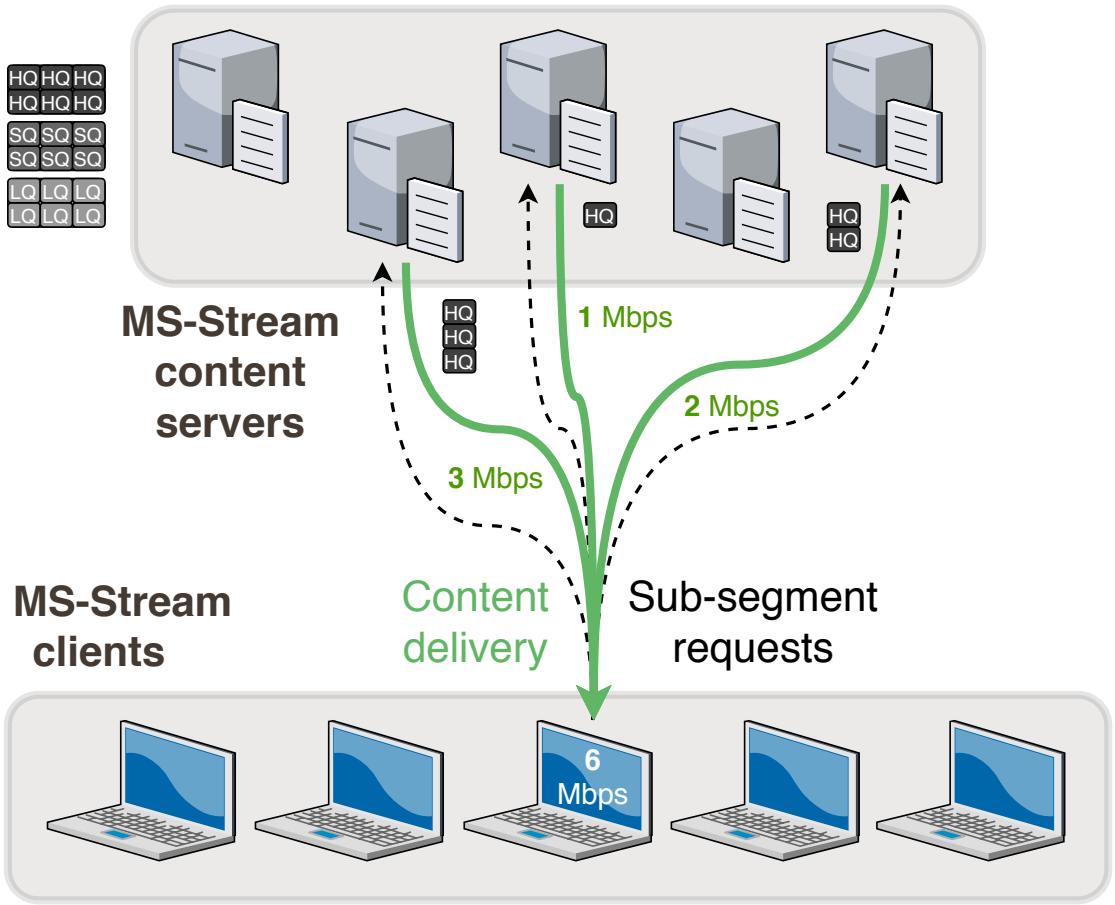


Figure 2.4: Multi-source adaptive streaming with MS-Stream

Let us take Figure 2.4 as an example and assume that the video segment is formed of 6 GoPs. The client asks the first server, with a capacity of 3 Mbps, for GoPs 1, 2 and 3 in HQ and GoPs 4, 5 and 6 in LQ. The client requests GoP 4 in HQ from the second server, with a capacity of 1 Mbps, and GoPs 1, 2, 3, 5 and 6 in LQ. Similarly, the third server sends GoPs 5 and 6 in HQ, and GoPs 1, 2, 3 and 4 in LQ. The client finally assembles a video segment with the highest received bitrate for each GoP and displays up to 6 Mbps.

Overhead

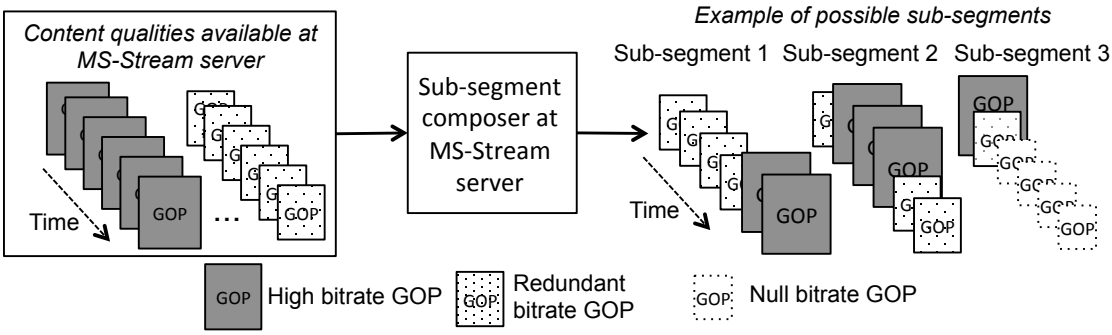


Figure 2.5: MS-Stream sub-segment generation and composition

In MS-Stream, the client uses the redundant GoPs in LQ as fallbacks if segments in HQ are not received on time. The bandwidth overhead directly depends on the bitrates and configuration used. Indeed, the MS-Stream client may be tuned to minimize the bandwidth consumption overhead resulting from GoPs redundancy by not requesting some of the LQ GoPs, as shown in Figure 2.5, sub-segment 3. This effectively results in a lower overhead. Results show less than 10% average increase of bandwidth usage in common usage of MS-Stream [BQLN+17c]. Besides, it ought to be noted that the generation and aggregation of sub-segments have very low processing footprints [BQLN+17c] as they only require an assembling function of already encoded GoPs available at different bitrates.

Summary

Nowadays, according to the DASH standard (and similar HAS techniques), video content is simultaneously encoded in various qualities, and clients request the most suitable quality according to network conditions.
MS-Stream is a multiple-source streaming solution that extends the DASH standard. MS-Stream clients can aggregate bandwidth from multiple servers to increase video quality and drastically reduce rebufferings, thanks to slight redundancy with low network overhead cost.
However, users can still experience server-side failures, congestion, or unavailability. Moreover, providers who statically over-provision their platform to mitigate these issues face a higher cost. Besides, browsing on video streaming platforms generates an access history of watched content, which 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.*, targeted advertising). Therefore, adaptive streaming still lacks a system providing both a high QoE and privacy protection to users while minimizing the underlying infrastructure cost.

Chapter 3

Related work

> *Outside of a dog, a book is man's best friend. Inside of a dog it's too dark to read.*
>
> — Groucho Marx

Video streaming is a trending topic both in research and in the industry, as consumers demand is continuously growing. This thesis aims at proposing a practical privacy-preserving video streaming system, providing both a high Quality of Experience and strong privacy guarantees to its users, without quality or performance degradation, at the lowest monetary cost. To achieve this goal, we propose three main contributions :

- `Muslin` is a dynamic server provisioning and advertising system which both increases QoE (with multiple-source streaming) and decreases costs (due to real-time delivery conditions monitoring).
- PrivaTube is a video streaming system intending to provide strong privacy guarantees (through the use of encryption and fake requests in Intel SGX enclaves to obfuscate access patterns), with unaltered QoE (thanks to edge-assistance and proactive pre-fetching using popularity-based caching policies).
- `PProx` complements video streaming systems by providing pseudonymous and private recommendations (being encrypted and shuffled in Intel SGX enclaves) with unaltered accuracy and high performance.

Therefore, in this chapter, we review related work in streaming platforms architectures, techniques for privacy-preserving streaming, and privacy-preserving recommender systems to complement video streaming solutions, which led to `Muslin`, PrivaTube and `PProx`.

3.1 Video streaming

Streaming platforms can target either live streaming (*i.e.*, the broadcast of a single stream, as it is generated) or VoD streaming. Then, to deliver content, these platforms can either rely on Peer-to-Peer (P2P) and/or hybrid (edge-assisted) delivery, meaning that end-users can retrieve content from other clients, or on CDN networks where content is exclusively delivered from video servers.

3.1.1 Edge-assisted and Peer-to-Peer streaming platforms

Live streaming does not require maintaining a cache for a collection of videos, making it adapted to decentralized streaming [LGL08]. Exchange of video segments may happen between peers over an unstructured network using gossip-style interactions, as in CoolStreaming [XLKZ07], or use dissemination structures built over an overlay network, as in MDC [PWC03] or SplitStream [CDK+03]. Pure non-hybrid 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 to 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 [SDK+07] 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 [TM12]. This is similar to the use of fake requests in PrivaTube but Push-to-Peer does not consider privacy aspects. P2VoD [DHT04] combines source servers with the use of multicast trees [PWC03, CDK+03]. Video content is cached at clients who can join the trees and act as providers. BASS [DLHC05] is a similar approach using the BitTorrent protocol. Both P2VoD and BASS are only evaluated using simulations, and the ability of the P2P network to provide high QoE remains limited. The approach proposed by Zhang, Wang, *et al.* [ZWCR09] is dual: A set of helper servers are used to complement a P2P VoD system when QoE or scaling cannot be guaranteed.

PrivaTube adopts an edge-assisted architecture where the discovery of 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 [ZLHC14] and LiveSky [YLZ+09].

Both systems combine CDNs with edge resources and report serving tens of millions of users simultaneously.

PeerTube [Peeb] is an open social network for video sharing. PeerTube is decentralized and uses a collection of support sites, leveraging W3C's federation protocol ActivityPub [W3C], but the download of videos is made using a single server.

3.1.2 WebRTC

WebRTC (Web Real-Time Communication) [Web], is an standardized API (by W3C and IETF) to communicate from one browser to another. The project started in 2011 and the API has been implemented in major browsers from 2013 to 2015. This technology have been a game changer for browser-based P2P use cases. The goal of WebRTC is to provide ultra-low latency UDP sessions in order to support real-time scenarios, such as video conferencing or multiplayer gaming between users. It embeds security and NAT-traversal functionalities. Its API is composed of three main parts: **User Media**, **RTCPeerConnection** and **Data Channels**.

User Media The User Media API offers tools to detect, select and use media equipment like cameras and microphones. With these functions, clients are able to create media streams from a webcam and send them to other peers. The media can be modified lively, which allows for instance to switch cameras of a cell phone. The packets produced by the stream can be caught in order to save the video or transfer data to an external server using any protocol.

RTCPeerConnection These functions aim to establish a connection between two peers. It enables providing servers for NAT-traversal capabilities, defining if a connection should be reliable, and if the packets should arrive in order. Then, a Session Description Protocol (SDP) string is created, containing information to create a link between peers. The SDP should be sent to the corresponding peer. A bidirectional websocket-based signaling server is usually leveraged to ease the initialization process. Once the SDP is sent, the browser begins the Interactive Connectivity Establishment (ICE) mechanism with the help of the STUN and TURN servers provided in the options. If a direct connection can be established with the help of a STUN server, this solution is prioritized. However, if the NAT can not allow such a link, a relay TURN server is selected to establish the session. Once the peer connection is ready, it becomes possible to send the media stream created with the **User Media** API to begin a simple ultra low latency video conference.

Data Channels Once a RTC connection is established, a data channel can be created (if both peers support data transmission). Raw data can be sent from one peer to another over UDP channels. This feature brings a lot of flexibility because any kind of information can be transferred. It may be used to transfer custom control flow, specific text messages, or even video segments. Because of these data channels, adding P2P capabilities to existing JavaScript DASH players has become possible. Instead of using a standard HTTP request to get a segment from a server, a custom query can be sent to another peer through a reliable channel. Proprietary solutions from Peer5 [Peea], Streamroot [Str] or Hive Streaming [Hiv] are using WebRTC data channels in their players. Similarly, the open source library P2P Media Loader [P2P] provides P2P capabilities on top of HAS video players thanks to WebRTC. The API also draws researchers attention to improve DASH segments delivery [ZLM16].

3.1.3 CDN-based streaming platforms architectures

In both live and VoD streaming, servers provisioning, video content replication and servers advertising are the key problematics for CDN operators. Most commercial CDN operators thus keep their policies secret [Pas12], as they often have a strong impact on cost and end-users' QoE. The usual paradigm is to estimate the audience for an event, and to provision enough servers near end-users to withstand the demand [Pas12]. Optimizing content replication is a difficult task. Replicating content to minimize the network distance for requesting clients is NP-complete [KRR02]. Therefore, advanced content caching algorithms are mostly heuristics. Then, when clients request video content, the CDN operators strategy is to route their requests to the nearest server thanks to DNS [NSS10] or IP anycast [FMM+15], and use HAS protocols for delivery. This behavior minimizes network-induced latency, and lowers the probability to encounter congestion.

Content replication policies

The most widespread content caching and replication techniques are based on greedy heuristic algorithms. It is usually done by maximizing a utility function [LOH16] or minimizing a cost function [SG16, LBSR14]. Other policies consider social relationships between users and forecast the trending videos [HWC+15]. Our work in `Muslin` is also based on a greedy iterative algorithm, however it differs from these propositions. Live content is only stored for a short time and requires fast computation and decision, as opposed to on-demand streaming where caching policies can converge over time. In PrivaTube, caching (pre-fetching) policies are probabilistic and take into account content

popularity over a long period, which is only possible in the VoD model. PrivaTube does not aim at forecasting future audience, or targeting specific locations.

Besides, some works use network awareness [BBC12] and QoS metrics to route requests or to select servers. Zheng *et al.* [ZT15] base their approach on path latency optimization through multiple servers, but not bandwidth or system scale. Similarly, Puntheeranurak *et al.* [PSn15] only take into account latency, delay and jitter inside the network. As opposed to these approaches, `Muslin` aims at reaching a high end-user QoE, and takes into account not only network measurements but also live clients feedbacks to provision servers.

Servers selection for a high and fairly shared QoE

Adhikari *et al.* [AGH+12] introduced the DASH framework of Netflix, the largest DASH provider worldwide, and outlined that a user is always bound to one server, regardless of network issues. Consequently, one major drawback is that servers can get overloaded, and thus some clients may receive a poor QoE or might even not have access to the content at all. `Muslin` takes into account not only the distance, but also the server bandwidth and requests failure (timeout) rate, enabling to provide a better QoE to the users. PrivaTube similarly computes latency, bandwidth and timeouts for each client to select Candidate Assisting Peers (CAPs) to fetch content from.

Besides, there have been some attempts to reach a better QoE fairness between HAS clients. Georgopoulos *et al.* [GEB+13] use Software Defined Networks to allocate bandwidth to each link, and Petrangeli *et al.* [PFC+15] adapt the video bitrate requested by clients. However, to the best of our knowledge, all approaches towards higher QoE fairness are single-source oriented, unlike `Muslin` and PrivaTube, and do not consider dynamically advertising servers to the clients as `Muslin` does.

3.2 Privacy-preserving streaming

As efficient video streaming over the Internet itself emerged recently, very few proposals were made to preserve users' privacy when using streaming platforms and solutions.

Popcorn [GCM+16] 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. The cost of PIR is related to the size of

the catalog. However, Popcorn has a weaker adversarial model than PrivaTube and `PProx` 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 `PProx`, and the use of lightweight cryptographic mechanisms make them scalable by design.

Rajan *et al.* [RVN+16] leverage a privacy-preserving publish and subscribe [OFMR16] 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 [DMS04]) this approach requires cryptographic operations at intermediary nodes, which incurs latencies and costs likely to degrade QoE significantly.

Cui *et al.* [CAR17] 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 and `PProx` do not require a specific and costly content encryption on servers.

3.2.1 Unlinkability-based solutions

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.* [DBYEV19] developed concurrently to our work a technique similar to our use of fake requests, but for pure P2P video streaming, *i.e.*, without any video servers. Peers subscribe to $k-1$ additional streams for each stream of interest, and participate in 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 [BSG17] could also benefit PrivaTube.

Other approaches target other applications such as P2P file sharing. For instance, Swarmscreen [CDM+09] 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, and similarly as in PrivaTube, improves overall content availability. 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 [dSDR16] leverages erasure codes to reduce the cost of fake requests in a P2P file-sharing system and, as for PrivaTube, improve content availability while concealing users' interests.

3.2.2 Designing privacy-preserving systems using Intel SGX

Trusted Execution Environments (TEEs) offer guarantees of isolation, confidentiality, and integrity of data and computations performed in untrusted machines by leveraging custom microprocessor zones. PrivaTube and `PProx` rely on Intel Software Guard Extensions (SGX) [CD16a], 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, meaning 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. Local attestations allow enclaves running on the same processor to prove each other as genuine, and remote attestations allow enclaves to prove their authenticity to third parties by making use of a remote attestation service provided by the manufacturer. 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.

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 [BWG+16], Web search proxies [MBF+17a, PGM+18] or database systems [SCF+15, PVC18]. This demonstrates that TEEs represent a promising technology as they offer a satisfactory compromise between security properties and performance overheads. While PrivaTube and `PProx` principles are not exclusively relying on Intel SGX properties, it still illustrates the benefits of this technology for performance-sensitive applications (such as video streaming).

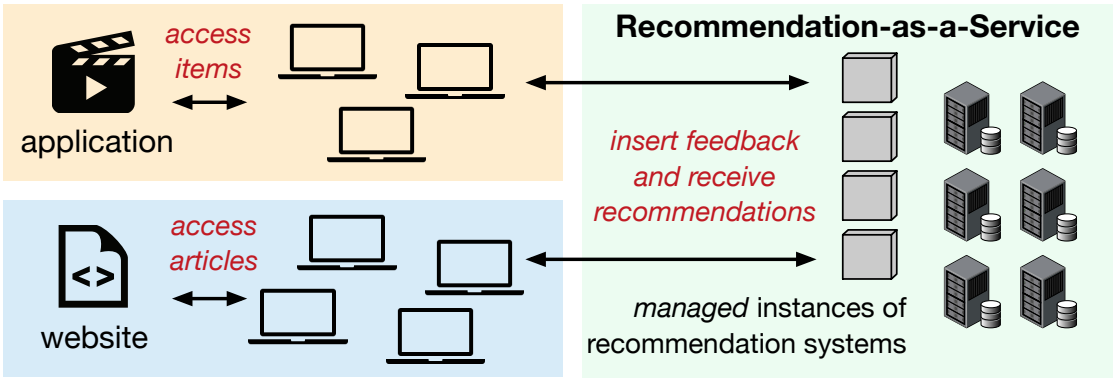


Figure 3.1: Principle of Recommendation-as-a-Service

3.3 Recommender systems

All major video streaming services include a recommendation functionality. They rely on these recommendations to retain users on their website or application. Indeed, recommender systems [BOHG13] complement traditional navigation on websites and applications. They enable personalized services [GFD+14], improve the user experience [GDBJ10], and eventually increase service providers' revenue [FH07]. Recommendations can be new directions, items, or media. These are computed based on users' past interactions and explicit or implicit feedback (*e.g.*, item likes, dislikes, navigation clicks) combined with the interactions of other users with similar interests. Recommendations are used by major Web services (*e.g.*, Google News, Amazon, Netflix) but can also benefit smaller players (*e.g.*, discussion forums or online stores).

3.3.1 Recommendation-as-a-Service

Configuring and operating an efficient and scalable recommendation service is far from trivial. Several companies thus offer Recommendation-as-a-Service (RaaS). Examples include Darwin & Goliath [BGO19], Mediego [Med19], Plista [Pli19b], or Recombee [Pli19a]. In the RaaS service model, illustrated in Figure 3.1, application providers delegate the collection of interaction information (*feedback*) from their users, the construction of models based on this feedback, and the generation of personalized item recommendations from their catalog.

3.3.2 Privacy issues for Recommendation-as-a-Service

The major downside of using recommendation systems is the impact on users' privacy. Computing recommendation requires, indeed, the collection of massive amounts of sensitive data, which raises legitimate concerns amongst users [VZ19]. Access histories and feedbacks may reveal personal traits or interests, *e.g.*, based on access to different topics in an online forum or specific movies in a review platform. An adversary observing interactions with the recommender system or accessing its database may succeed in profiling users and determining private information such as their faith, sexual preferences, or health condition [CKN+11].

Privacy risks of recommender systems can be, unfortunately, amplified by the use of RaaS. Applications now have to trust a third-party and its infrastructure, typically running in a public cloud, for receiving and storing sensitive data from their users.

3.3.3 Privacy-preserving Recommendation-as-a-Service

The research community proposed various solutions to deal with the privacy concerns of recommender systems. These solutions can be classified in three categories: (i) those based on cryptography [WTAR19, GKP+17] where computations are performed over fully-encrypted data; (ii) differentially private solutions [MM09, SKSX18, SJ14] that add noise for disallowing the re-identification of a specific user and their data and (iii) Peer-to-Peer solutions [BFG+16, DTS+19, CA07, CC02, SPTH09] where users keep their preferences locally and compute in a decentralized manner their similarity with other users based on commonly accessed items. These solutions present drawbacks such as performance issues for cryptography-based solutions or accuracy issues for differentially private solutions (due to the addition of noise) and P2P solutions (due to the partial knowledge users have on the overall system). Perhaps more importantly, none is well adapted to the RaaS service model, and that for two key reasons:

- They all target a *specific* type of recommendation algorithm (*e.g.*, using matrix factorization [SKSX18, DTS+19] or collaborative filtering [CC02, SPTH09]). This goes against the need for RaaS providers to support a variety of such algorithms [Bur02, BOHG13];
- They require to install complex code and to maintain specific or even sensitive information at the user-side, at odds with the "turn-key" service model of RaaS. Solutions based on encrypted processing [GKP+17, WTAR19] require to provision

secret keys to the user side with the associated risks of leakage and the additional complexity of large-scale private key management. Differentially private solutions implemented at the client side [SJ14, MM09] require to provide clients with models of the data domain to enable adaptive noise addition.

Privacy violations related to recommendation received considerable attention in research [BOHG13, FKV+15]. Representative risks are the inference of individual users' profiles from temporal changes in the public outputs of a recommender system [CKN+11], or statistical de-anonymization attacks [NS08]. Surveyed users generally consider that recommendation systems violate their privacy [MDMÖG18] and would prefer not to be profiled [AK06].

Privacy preservation can involve cryptographic schemes such as homomorphic encryption to compute recommendations over encrypted user preferences, *e.g.*, using X-Rec [GKP+17] or CryptoRec [WTAR19]. These solutions have a high computational overhead, leading to high latencies in collecting recommendations. Slope One predictors [BVKD11] evaluations using support for homomorphic computations of the Paillier cryptosystem [Pai99] report, indeed, latencies in the order of several seconds in public clouds [BVK+12, BVKD13], similarly as for CryptoRec [WTAR19]. `PProx` only imposes a limited latency on top of the base performance of an unmodified recommender system.

Differential privacy limits the disclosure of private information of records in the result of aggregate queries in a statistical database [Dwo08]. In the context of recommender systems, differential privacy can be used to add noise and obfuscate user preferences in the LRS storage and replies [MM09, SJ14, SKSX18]. Such approaches come with a difficult-to-set trade-off on the quality of recommendations. Under our fault model, the noise should further be added before sending put requests to the cloud, requiring the provision of user-side code with specific models. In contrast, `PProx` does not degrade the quality of recommendations and enables easy deployment.

A final approach to privacy preservation is to distribute the computation. PDMFRec [DTS+19] enables decentralized matrix factorization, an operation at the heart of many recommendation algorithms. Other approaches include the pre-aggregation of several users' profiles and the use aggregated profiles in the cloud [SPTH09], or P2P approaches computing an overlay of nodes based on similar interests [BEK16]. Data decentralization reduces risks of leaks in the cloud but increases such risks during direct exchanges between users. These solutions have to rely on additional noise to protect individual profiles, impacting the quality of recommendations, and their deployment is far from trivial (*e.g.*, considering NATs, firewalls, or the possibility of malware).

X-Search [MBF+17b] implements web search proxies in SGX to protect the link between users and their search queries. While this presents similarities with user-interest unlinkability, X-Search employs fake queries to obfuscate this information. Such an approach would not be applicable to a recommender system as it would degrade the quality of recommendations. SGX-Tor [KHH+18] leverages SGX to strengthen the security of Tor. Fake requests in PrivaTube and shuffling in `PProx` present similarities with onion routing in Tor, in that they help prevent an adversary observing network exchanges from determining communication endpoints. Unlike `PProx`, and similarly to other work employing SGX [KCG17, KPW+19], X-Search or SGX-Tor do not consider the possibility for an adversary to steal secrets from an enclave.

State of the Art overview

To sum up this chapter, Table 3.1 provides the reader with an overview of streaming platforms architectures, techniques for privacy-preserving streaming, and privacy-preserving recommender systems.

Category	Examples	Comments	Contribution
	– Traditional streaming –		
CDN+HAS Decentralized	YouTube [You20] Dailymotion [Dai20] Twitch [Twi20] Vimeo [Vim20] Netflix [Net20] PeerTube [Peeb]	Clients are bound to a single server, thus prone to failures, congestion or unavailability, and therefore unfairness.	`Muslin` dynamically provisions and advertises content servers to users, and simultaneously streams from multiple sources.
	– Peer-to-Peer –		
Full P2P	CoolStreaming [XLKZ07] MDC [PWC03] SplitStream [CDK+03]	P2P provides scalability by essence but is unstable and unreliable.	`Muslin` & PrivaTube ensure reliability with proactive provisioning.
	– Edge-assisted –		
WebRTC	Peer5 [Peea] Streamroot [Str] Hive Streaming [Hiv] P2P Media Loader [P2P]	Edge-assisted streaming delivers a higher QoE than pure P2P streaming while remaining scalable. However, it still lacks privacy preservation, as peers and providers can access sensitive data.	PrivaTube provides strong privacy guarantees through the use of encryption and fake requests in Intel SGX enclaves, with unaltered QoE thanks to edge-assistance and multiple-source streaming.
Classic	Push-to-Peer [SDK+07] P2VoD [DHT04] BASS [DLHC05] Xunlei Kankan [ZLHC14] LiveSky [YLZ+09]		
	– Privacy-preserving streaming –		
General	Popcorn [GCM+16] Tor [DMS04]	High latency overhead and scalability issues.	PrivaTube scales well without altering QoE.
Unlinkability	P3LS [DBYEV19] Swarmscreen [CDM+09] Mistrustful P2P [dSDR16]	These techniques have important bandwidth and storage overheads.	PrivaTube further leverages fake requests for proactive caching.
	– Privacy-preserving recommenders –		
Cryptography	X-Rec [GKP+17] CryptoRec [WTAR19]	Extremely slow, often several seconds latency.	`PProx` provides sub-second latencies.
Differential Decentralized	[MM09, SJ14, SKSX18] PDMFRec [DTS+19]	Quite inaccurate, and impractical due to complex client-side code.	`PProx` preserves accuracy with minimal deployment requirements.

Table 3.1: State of the Art digest

Summary

Current video streaming solutions usually rely on CDN servers and HAS techniques to deliver content. However, the client is often bound to a single server, thus prone to failures, congestion or unavailability, and therefore unfairness between users. Besides, providers who statically over-provision their platform to mitigate these issues face a higher cost. To overcome these challenges, `Muslin` dynamically provisions and advertises content servers to users and simultaneously streams from multiple sources.

An alternative to CDN servers is P2P streaming, which provides better scalability by essence, but is unstable and unreliable. Thanks to enabling solutions such as WebRTC, hybrid edge-assisted streaming is currently growing. It delivers a higher QoE than P2P streaming, in a scalable fashion, such as in PrivaTube. However, it still lacks privacy preservation, as peers and providers can access sensitive data.

Very few privacy-preserving video streaming systems exist. Most of them rely on heavy cryptographic mechanisms over multiple nodes, adding a performance and latency overhead, effectively reducing QoE. Unlinkability-based techniques (*i.e.*, fake requests) are efficient but come at the cost of bandwidth and storage overheads. Therefore, PrivaTube relies on fake requests, but also leverages them for pre-fetching and caching, thus reducing their cost. Besides, the use of Intel SGX enclaves enables strong security properties with great performance (*e.g.*, encryption mechanisms in PrivaTube and `PProx`).

Recommender systems usually complement streaming solutions to retain users on their website or application. Yet, they pose a serious threat to privacy, as user profiles are established based on the watching history. A few privacy-preserving recommender systems exist. They can be either cryptography-based, usually very slow (several seconds latency); differentially private with added noise, thus inaccurate; or P2P-based and decentralized, often slow, less accurate and unreliable. Besides, all of them use specific recommendation algorithms, and require to install a heavy code layer at the client side. To tackle these issues, `PProx` combines a several-layer encryption mechanism with obfuscation (through shuffling) inside high-performance Intel SGX-enabled HTTP proxies.

Chapter 4

`Muslin`: High-QoE cost-efficient multi-source streaming

The way to get started is to quit talking and begin doing.

— Walt Disney

Streaming services usually rely on large-scale CDN infrastructures to host video content, and on HAS solutions (such as the DASH standard) to deliver it. When accessing a video stream, consuming clients are automatically re-directed to the closest server to temper network congestion and achieve higher throughput. However, if a large amount of end-users located under the same geographic area is simultaneously consuming the same streamed content, the nearest server can rapidly become overloaded.

This chapter introduces `Muslin` [DSBQL$^+$19, DSBQL$^+$18a, DSBQL$^+$18b], a solution supporting a high, fairly shared end-users QoE for live video streaming services over the Internet.

4.1 Introduction

Current video streaming solutions usually rely on CDN servers and HAS techniques to deliver content. However, the client is often bound to a single server, thus prone to failures, congestion or unavailability, and therefore unfairness between users. Besides, providers who statically over-provision their platform to mitigate these issues face a higher cost.

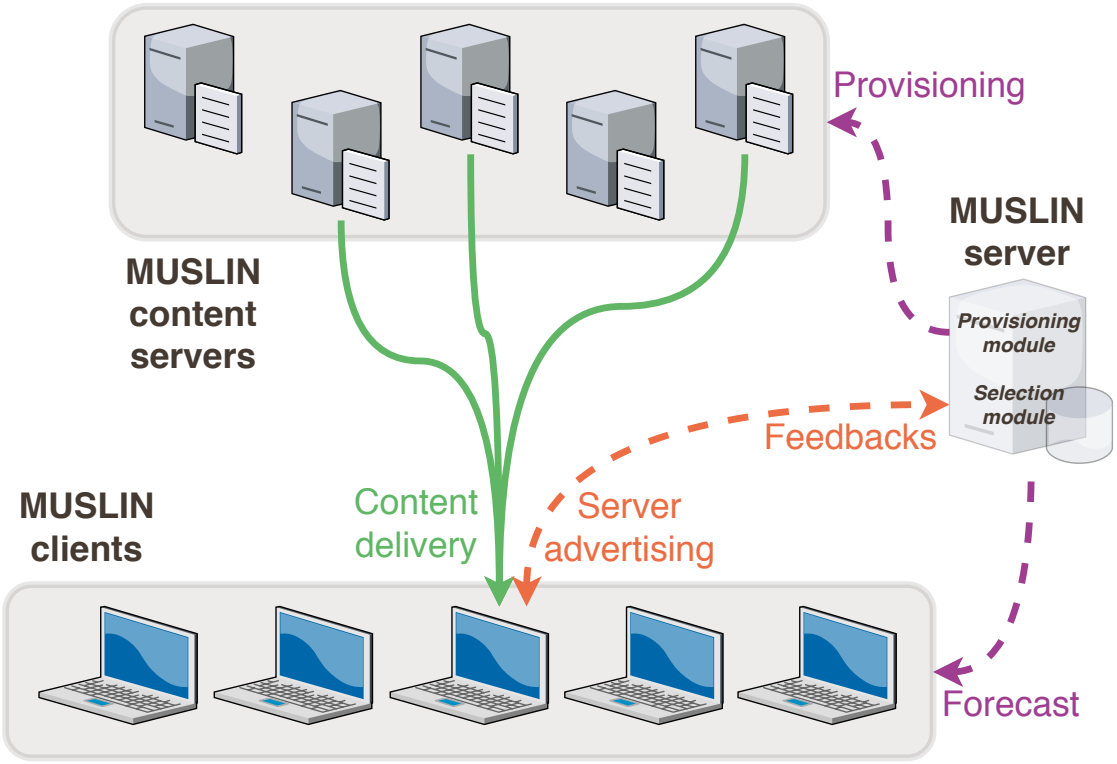


Figure 4.1: Muslin overview

To overcome these challenges, Muslin relies on periodic feedbacks from Muslin clients during streaming sessions and on a ranking score for servers provisioning and advertising. As shown on Figure 4.1, the Muslin server provisioning module periodically estimates the required throughput to dynamically adjust the infrastructure scale according to real-world needs. The Muslin server selection module then advertises relevant content servers to clients depending on multiple criteria such as distance, bandwidth and server load. For content delivery, Muslin leverages MS-Stream, a multiple-source streaming solution based on the DASH standard, in which a client can simultaneously use several servers to aggregate throughput from multiple channels and to offer a higher QoE for its users (see Section 2.2.3).

The rest of this chapter is organized as follows. Section 4.2 describes the Muslin solution and introduces the provisioning and selection modules. We detail our experimental setup in Section 4.3 and present our evaluation results in Section 4.4. Finally, Section 4.5 concludes.

4.2 Muslin: Multi-Source Live Streaming

Muslin's goal is to provide a high and fairly shared QoE for live video content delivery. As QoE is subjective, it is a difficult challenge to evaluate the QoE of end-users. QoE depends on many criteria, such as stalls, video resolution, encoding quality factor, bitrate fluctuation over time, glitches, etc. The ITU-T recently provided automated methods to algorithmically assess streaming QoE according to multiple factors in the P.1203 recommendation [IT17]. As it is complex and costly to take all parameters into account, Muslin tackles the main reasons why end-users are not satisfied with their streaming experience, which are the number of rebuffering events, the average video bitrate displayed, and the number of quality changes during the session. Indeed, rebuffering events are considered the main negative impact on perceived QoE [HSH+11], and both the average video bitrate and the quality changes have a significantly higher influence on QoE in adaptive streaming than other criteria [SES+14].

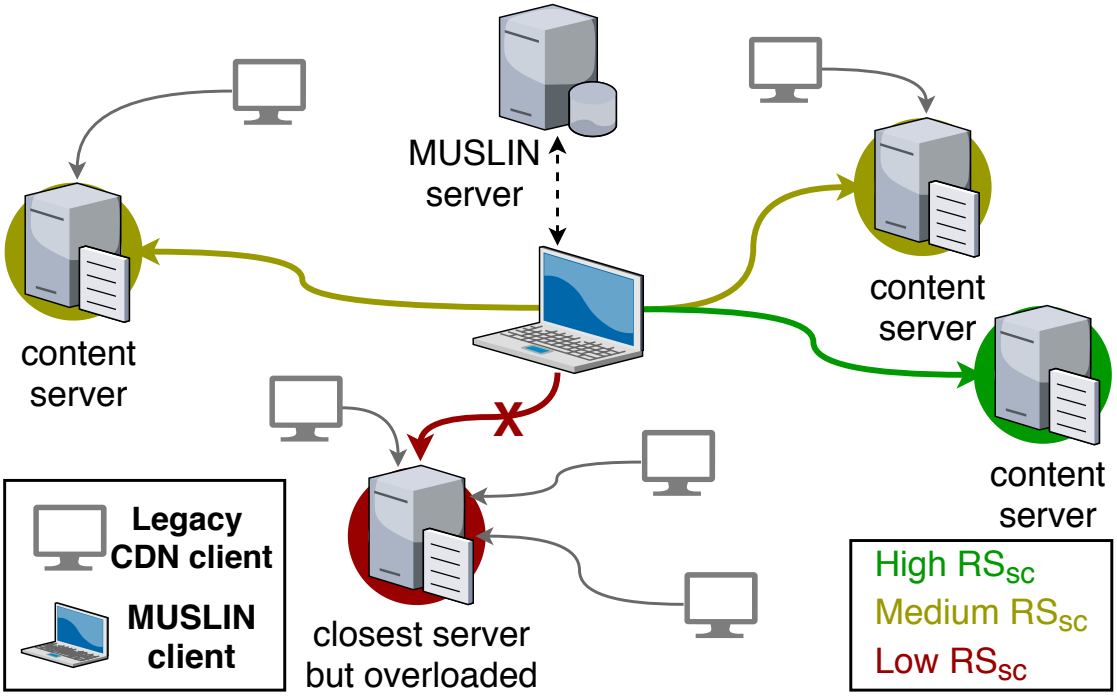


Figure 4.2: If nearby content servers are overloaded, the Muslin server selects and advertises other content servers with a higher Ranking Score RS_{sc} to the client.

Muslin intends to solve the root causes for such QoE degradation, the two main reasons being (1) the server load and (2) the low bandwidth between the server and the client. Indeed, if a server is overloaded or if the network channel bandwidth to this server

is low, clients requests to this server will timeout and cause rebufferings or visual quality degradation (as shown on Figure 4.2). Therefore, Muslin is able to monitor current delivery conditions to adapt its delivery schemes.

The Muslin system is composed of a Muslin server, MS-Stream clients, and MS-Stream content delivery servers with an additional Muslin layer to handle feedbacks and provisioning. Muslin clients send periodic feedbacks to the Muslin server, including the observed bandwidth from each server, the video sub-segment requests failure (timeout) rate, their average displayed video bitrate, the number of rebufferings they experience, and the number of quality changes. Then, based on these feedbacks, the Muslin server accordingly scales the underlying delivery platform to provide a higher QoE to end-users.

Fairness among users is mostly achieved thanks to the periodic feedbacks sent from the clients. They aim at monitoring the QoS and QoE each user is provided with, and improve server provisioning and selection accordingly. Server and Network Assisted DASH (SAND)[DI18], introduced in MPEG-DASH Part 5, defines a standard for control messages exchanged between the servers and clients to report metrics. Muslin feedback messages are currently not compliant with SAND, as Muslin was developed prior to this standard, but will be in a future version for better interoperability. Besides, MS-Stream allows to maximize server throughput and reduce competition between clients when CDN servers resources are saturated, as each client depends on multiple servers and is not bound to a specific one.

Muslin is specifically effective for live streaming where churn rate can be very high and important audience fluctuations can happen within seconds. In a VoD use case, clients' buffers can be larger and there is less pressure to react in real-time.

As illustrated in Figure 4.3, (**1**) the Muslin server dynamically provisions content servers and replicates content to available MS-Stream content delivery servers, which then register themselves to the selection module; (**2**) when a client requests a MPD file, the selection module replies with a list of available servers; (**3**) the client can access live content and begin the streaming session with the MS-Stream protocol; (**4**) Muslin clients send periodic feedbacks. In this section, we present in details the Muslin system and the Muslin server two main components, the provisioning module and the selection module.

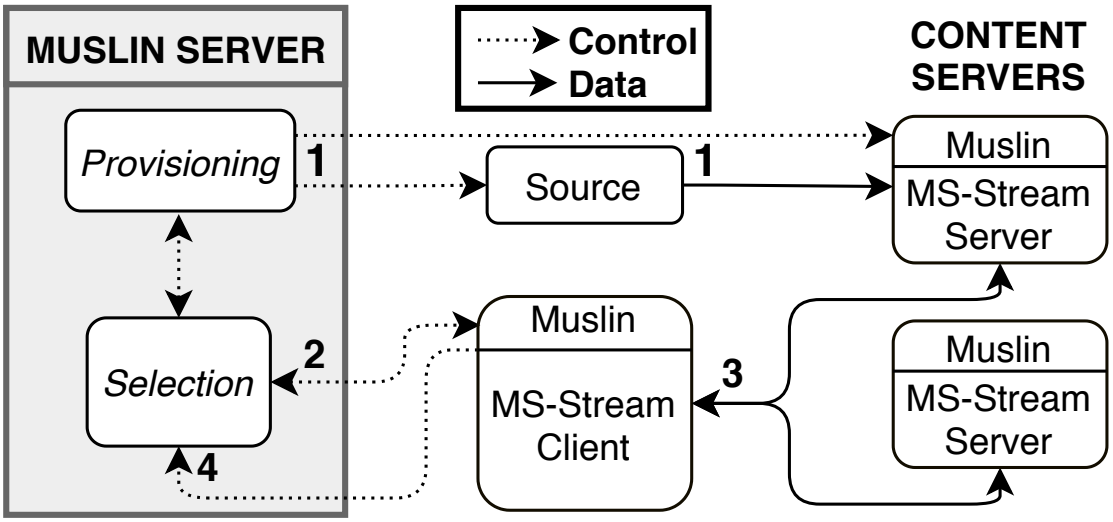


Figure 4.3: `Muslin` system architecture overview

4.2.1 Provisioning module

The provisioning module goal is to decide on the number of servers to provision not only to answer end-users throughput demand in video contents, but also to maximize their QoE and minimize the required infrastructure scale. To do so, it periodically estimates the required throughput to fulfill the demand based on actual feedbacks, and provisions a subset of servers to host the content. The provisioning module period T is equal to the length of two segments (typically 10 seconds).

Audience forecast

In order to estimate the demand, `Muslin` computes the future number of clients during each period T. The current audience is defined as v_t. The estimated audience at the next iteration $(t + T)$ is labeled $\widehat{v_{t+T}}$. Finally, Δv represents the change in number of viewers, that is to say $\Delta v = v_t - v_{t-T}$. `Muslin` estimates the audience with the following formula:

$$\widehat{v_{t+T}} = v_t + \Delta v \tag{4.1}$$

As the actual replication is mostly based on clients feedbacks, a more accurate estimation is not required.

Throughput estimation

`Muslin` throughput estimation algorithm uses the demand forecast $\widehat{v_{t+T}}$ to estimate how much throughput D the overall system must provide to the users. Each client tries to reach a target quality (highest available video bitrate) Q. Due to MS-Stream specification, the sub-segments redundancy adds a network bandwidth overhead percentage O (up to a user-defined parameter). Besides, we introduce C, a dynamic corrective coefficient to address the network and server issues. It takes into account the mean average video bitrate B ($B \leq Q$) displayed by all clients watching the stream, and the failure rate FR which is the proportion of clients who failed to obtain in time the response of their last request from the server.

$$C = \frac{Q}{B} * (1 + FR) \tag{4.2}$$

The dynamic coefficient C allows the system to scale according to current clients QoE. It is then possible to compute the required system throughput that will be requested by the clients, using the following formula:

$$D = C * \widehat{v_{t+T}} * (Q + O) \tag{4.3}$$

Provisioning decision

The provisioning module decides which servers to provision. To do so, the provisioning module periodically computes a server Ranking Score RS_s for each server s (including offline servers), based on clients and servers proximity and on feedbacks gathered periodically from all clients:

$$RS_s = (N_s * (1 - FR_s) * OBW_s)^{\frac{1}{3}} \tag{4.4}$$

As shown in Equation 4.4, the RS_s takes into account the number of nearby clients N_s, the failure rate FR_s, and the average observed bandwidth OBW_s for each server s by computing a geometric mean. The higher the score, the more likely the server to be provisioned. For each server, the number of clients for which this would be the closest

content server is computed as N_s. `Muslin` clients report when servers fail to deliver a sub-segment in time. This measurement is aggregated into the failure rate FR_s. It represents the ratio of delivery failures detected over the total number of clients that requested a sub-segment from this server during the last T seconds. Besides, all clients can estimate the bandwidth from a specific server by observing delivered throughput in past requests. `Muslin` can compute the average observed bandwidth estimate OBW_s for each server s.

First, the RS_s of content servers is computed, and they are sorted by decreasing order. If the target throughput D is greater than the current system maximum available throughput, more servers are iteratively provisioned (by descending RS_s order) until D is reached. Else, if the system is over-provisioned, the servers are deprovisioned according to their RS_s in ascending order.

4.2.2 Selection module

The `Muslin` selection module goal is to advertise a subset of available content servers to each client. To this end, we define a client-specific Ranking Score RS_{sc}, in order to reach a high and fairly shared QoE:

$$RS_{sc} = ((D_{max} - GD_{sc}) * (1 - FR_s) * OBW_s)^{\frac{1}{3}} \tag{4.5}$$

To order the list of available content servers, the selection module computes the RS_{sc} for all server s and client c, based on feedbacks periodically sent by `Muslin` clients during streaming sessions. Similarly to the provisioning score, the RS_{sc} is based on the distance between each client and server, and on clients feedbacks. As shown in Equation 4.5, the client-specific ranking score includes the maximum distance between any two places on Earth (D_{max} kilometers, roughly 20000), the geographical distance GD_{sc} using geoIP data inferred from IP addresses, the video sub-segment delivery failure rate FR_s of server s (*i.e.*, the percentage of requests the server was not able to handle on time), and the average observed bandwidth OBW_s between all clients and server s.

When clients request a video content, the selection module returns a MPD file containing servers sorted by descending RS_{sc} order. Then, `Muslin` clients decide how many servers they use, based on MS-Stream adaptation strategies. As illustrated in Figure 4.4, if nearby content servers are already overloaded, the `Muslin` server selects and advertises other content servers with a higher RS_{sc} to the client. It prevents content starvation from

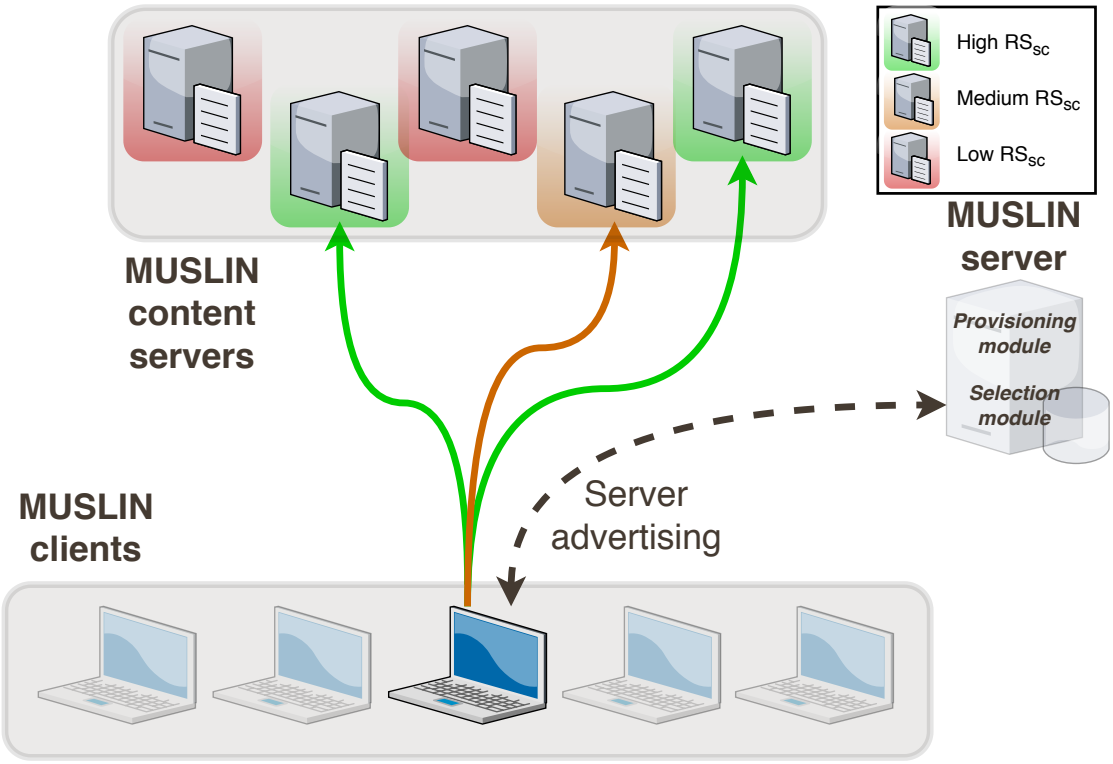


Figure 4.4: `Muslin` RS_{sc}-based servers selection example

clients, and allows fairness among users independently from their geographic position or nearby servers.

4.2.3 Implementation and scalability discussion

The `Muslin` modules and `Muslin` content servers overlay are implemented in Java and run inside light-weight Docker containers. `Muslin` content servers are built on top of MS-Stream servers by adding the necessary glue code to manage the interaction with the `Muslin` provisioning and selection modules. All interactions with the `Muslin` modules fulfill the REST architecture style. `Muslin` clients are developed in pure JavaScript and run within any mobile or desktop Web browser. Clients extend MS-Stream clients by featuring periodic feedback reports to the `Muslin` server.

In terms of scalability issues, the `Muslin` system scales similarly to current HAS solutions as MS-Stream is compliant with the DASH standard. A scalability downside is due to the periodic clients' feedbacks as the `Muslin` server workload grows linearly

with the number of clients. To solve this issue, we implement on the client a feedback request probability Pr to bound the number of feedbacks (see Equation 4.6).

$$\Pr = \min(1, N/v_t) \tag{4.6}$$

We thus ensure statistically that at most N clients will send a feedback for every period T, depending on the current audience v_t. With fewer feedbacks from the clients, the average estimated bandwidth and failure rates for servers are still correct but refreshed at a lower rate, resulting in temporary drops of QoE for some users.

Another scalability downside is due to the MPD refresh requests from `Muslin` clients every few segments, or when they experience a poor QoE. Similarly to the clients feedbacks, the `Muslin` server can become overloaded when too many clients request a new MPD file. To solve this issue, the `Muslin` selection module is distributed across several network nodes, each node only handling nearby clients requests (routed using classic DNS-based schemes).

4.3 Experimental setup

In order to evaluate our approach, `Muslin` was deployed and compared with various strategies that are commonly used. In the remainder of this section, we describe in details each implemented strategy and then present the testbeds and the audience trace we use for our experiments.

4.3.1 Provisioning, forecast, advertising and delivery policies

To evaluate `Muslin`, we implemented several common and alternative strategies as summarized in Table 4.1.

Table 4.1: Provisioning, audience forecast, selection policies, and delivery protocols

Provisioning	Forecast	Selection	Protocol
`Muslin`	(`Muslin`)	`Muslin`	MS-Stream
Geographical	Estimate	CDN	DASH
Random	Oracle	Random	
		Round robin	

Provisioning

Although CDN operators keep their strategies secret, the usual paradigm is to replicate content near end-users and to balance the load across multiple servers. Therefore, we implement two provisioning policies: *geographical* and *random*. The *geographical* policy is aware of the clients locations and replicates the content to servers near locations with the most clients. The *random* policy replicates content to randomly selected servers.

Audience forecast

Usually, CDN operators try to estimate the audience for an event, and then provision enough servers near end-users in advance to withstand the demand. Therefore, we implement an *oracle* forecast, which is aware of the exact amount of viewers and their locations at any time. This policy is of course unreachable in real life, but it provides a best-case current paradigm comparison. On the contrary, in the *estimate* strategy, the audience is periodically estimated with the strategy depicted in Equation 4.1.

Selection policy

We implement three selection policies called *CDN*, *Random* and *Round robin*. The *CDN* strategy is the most widespread one. It consists in routing clients to the nearest provisioned servers. In the *Random* policy, servers in the MPD file are randomly selected and sorted. The *Round robin* policy balances the load among available servers, as servers within the MPD file are permuted for each new client request.

Content delivery

To deliver video content, we used the multi-source MS-Stream solution and the single-source DASH standard.

4.3.2 Servers and clients setup

We evaluate our proposal in an actual environment. To do so, we set up 19 servers and 60 clients in our testbeds according to the US map (see Figure 4.5). We also chose an actual audience trace to generate clients churn.

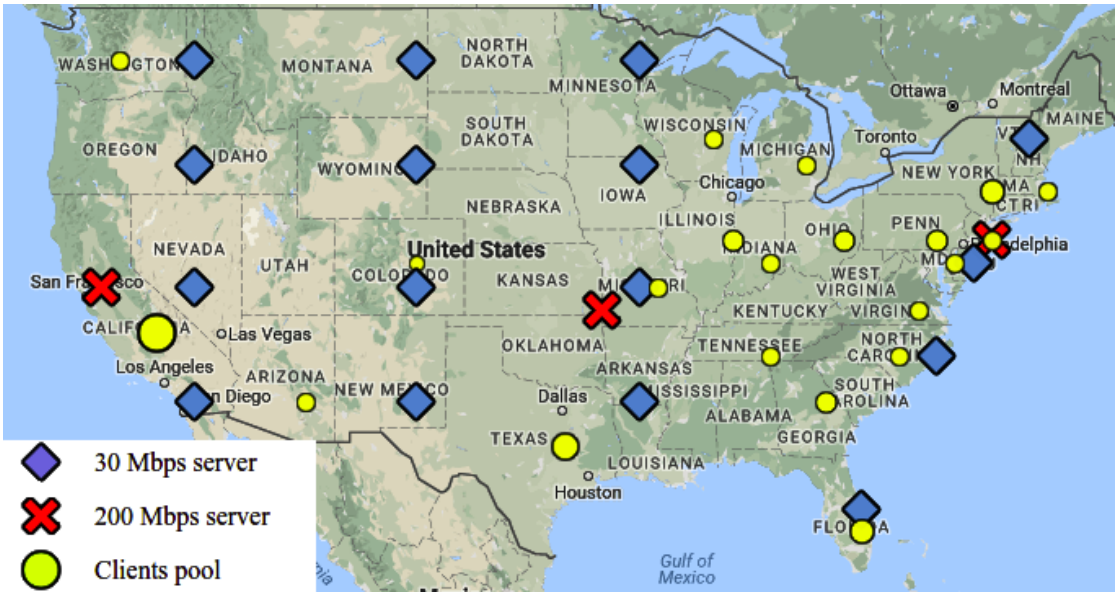


Figure 4.5: US map with points of presence and clients

Table 4.2: Available servers for each setup

ID	Location	Upload (Mbps)
3	California, USA	200
3	Kansas, USA	200
3	New York, USA	200
16	16 states	30

testbeds

As shown in Table 4.2, we set up multiple Points of Presence (PoP) geographically distributed in the US on a local network, by computing the latency and bandwidth between each client and server according to the geographical distance. Those PoP are setup according to two testbeds. In the first testbed, 200 Mbps servers are available in *3* strategic locations (West, center and East). In the second testbed, we use 30 Mbps servers located in *16* US states. We chose 16 locations as most CDN providers have between 10 and 30 PoP [CDN18], and Google provides 16 locations [Goo18].

Besides, we selected 21 client pools locations in the contiguous US states. We randomly distributed the clients in the states using a weighted probability matching the state population (*e.g.*, California: 13%, Texas 10%, etc.) as shown in Figure 4.5, and saved the output to re-use the same toss in all the experiments.

Audience trace

In order to be consistent for all experiments runs, we selected an audience trace and replayed it every time by automatically connecting or removing video clients from the broadcast, thanks to Docker containers.

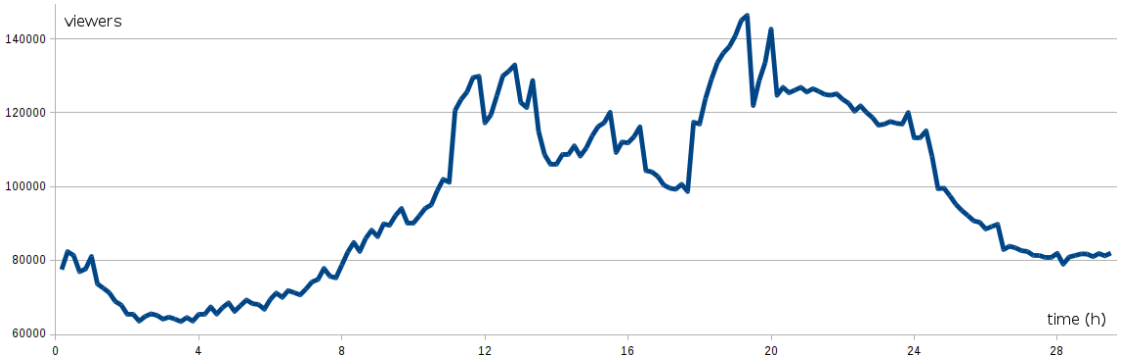


Figure 4.6: AGDQ audience trace

The audience profile we chose (Figure 4.6) is a real trace from a week-long charitable videogames event streamed online. The audience used is from July 08 2016 [Twi18], as it contains many typical audience patterns, from 60 000 to 150 000 viewers over 30 hours. We scaled down the number of simultaneous clients to 60 (about 250 unique sessions throughout each experiment) as our experimental infrastructure could not support hundreds of thousands of connections. All clients are desktop with 30 seconds maximum buffer and 8 Mbps download bandwidth.

Experiments

We perform our experiments using the `Muslin` system as described in Section 4.2 and the policies explained above. Our experiments consist in a 30 minutes live streaming broadcast, re-run 5 times to aggregate results and reduce noise and outliers impact in the distributions. The used live video content is the Blender Big Buck Bunny video encoded in five video bitrates (see Table 4.3).

4.4 Evaluation results

In this section we evaluate the delivery solutions and various policies in terms of cost, QoE and fairness.

Table 4.3: Available video qualities

ID	Bitrate (bps)
0	205.129
1	1.012.240
2	2.029.450
3	4.086.016
4	6.391.489

4.4.1 Delivery solutions

In this subsection we evaluate MS-Stream against single-source DASH streaming. Figure 4.7 represents the number of rebufferings for each evaluated combination. The X-axes represent experiment setups with IDs in the following form: *provisioning + forecast + servers selection + testbed + protocol*. For instance, *geographical oracle* replication with *random* servers selection on testbed *16* with MS-Stream protocol has ID *gor16m*.

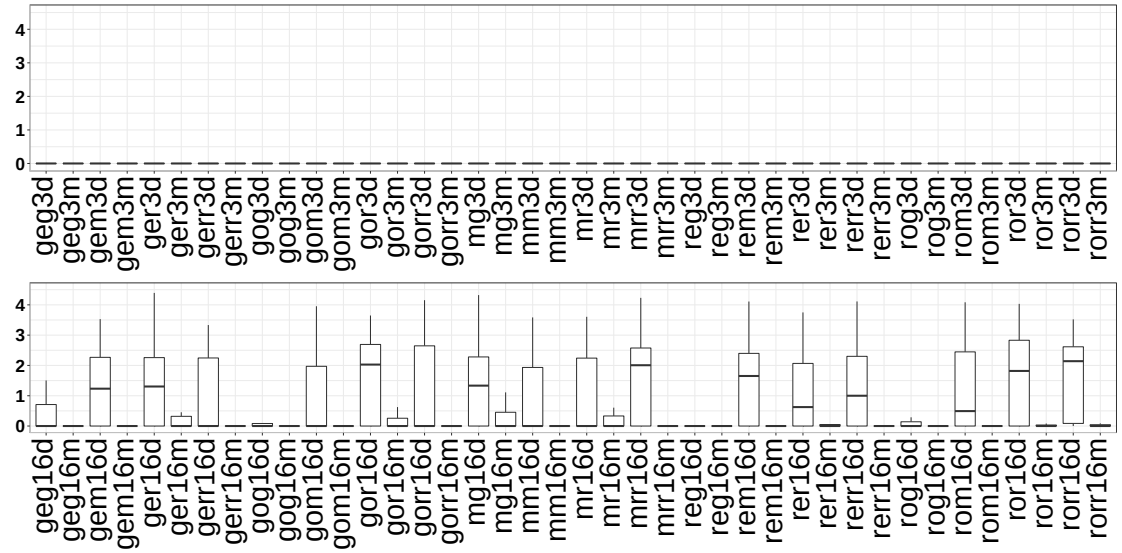


Figure 4.7: Number of rebufferings (per minute), 3 servers testbed (top), 16 servers testbed (bottom)

The 3 servers testbed provides all clients with a rebuffering-free streaming session, due to the high servers capacities and capabilities. In the 16-server testbed, the *CDN* experiment performed with the DASH standard instead of MS-Stream results in half the clients suffering at least one rebuffering in their streaming session. But when using MS-Stream instead of DASH, all the clients have a continuous playback. More generally,

every MS-Stream setup outperforms its DASH equivalent in terms of rebufferings, as most MS-Stream clients do not experience any rebuffering event at all.

In terms of bitrate, the results are quite similar between the two solutions in the 3 servers testbed. In the 16 servers testbed, MS-Stream provides a mean bitrate increase up to 4 Mbps over DASH and lowers the number of quality changes for all setups. Besides, the *CDN* setup with MS-Stream provides a more homogeneous distribution as all clients reach a quality higher than 6.2 mbps in both testbeds. Oppositely, more than a quarter of DASH clients display less than 6.0 mbps in the 16 servers testbed. Finally, the gains and losses in the number of quality changes for the 3 servers testbed varies with no distinguishable global trend. In the 16 servers testbed, all clients experience less quality changes when using MS-Stream, with up to 4 less quality changes per minute.

All these results are explained as MS-Stream was mainly designed to increase the end-user's perceived QoE by avoiding rebufferings, providing a smoother playback and simultaneously utilizing the available bandwidth from multiple paths with heterogeneous characteristics, as previously mentioned [BQLN+17c]. The downside of these QoE improvements is a small CPU overhead, and the compulsory network bandwidth overhead induced by the MS-Stream solution (evaluated in Section 4.4.5), which corresponds to the percentage of data downloaded by the client but not used. Further details and additional MS-Stream evaluations are available in former works [BQLN17a, BQLN+18, BQLN+17b, BQLN+17c]. In the rest of this chapter, we only consider the MS-Stream delivery solution, as it provides a greater QoE to the end-users for a small CPU and bandwidth overhead.

4.4.2 Provisioning cost

`Muslin` aims at providing a high and fairly shared QoE through multi-source live streaming, but it also aims at doing so while being cost-efficient when provisioning servers. To compute provisioning cost, we assume a cloud computing service using server time billing. Therefore, we sum the provisioned server time for each experiment run and compute relative values.

As shown in Table 4.4, the total server time required is lower when using audience estimates and dynamic server provisioning policies. Furthermore, as `Muslin` replication policy also takes into account the actual quality displayed by the clients when dimensioning the delivery system, it doesn't over-provision if all clients can already obtain the target video quality, and effectively lowers the number of servers provisioned when possible.

Table 4.4: Total relative cost (server time), 16 servers testbed

Provisioning	Forecast	Server time
`Muslin`	(`Muslin`)	100
Geographical	Estimate	109
Geographical	Oracle	118
Random	Estimate	109
Random	Oracle	118

In the 3-server testbed, all policies have roughly the same cost.

`Muslin` replication is thus the least costly policy for the 16-server testbed, as it allows more flexibility.

For better readability, we identify four relevant combinations, referred to as *Muslin*, *CDN*, *Random* and *Round robin* in the text, detailed in Table 4.5. We chose the *geographical oracle* provisioning and forecast combination because even though it is impossible to reach in real-life, it provides a best-case current paradigm comparison with `Muslin`. Likewise, we chose the *CDN*, *Random* and *Round robin* servers selection policies as they are the three most widespread load balancing strategies. Furthermore, as explained in Section 4.4.1, MS-Stream provides a greater QoE to the end-users than DASH, so we chose to only consider the former delivery solution.

Table 4.5: Selected provisioning, forecast, selection, delivery policies, and testbed

Name	Provisioning	Forecast	Selection	Delivery	Testbed
Muslin	`Muslin`	(`Muslin`)	`Muslin`	MS-Stream	16 servers
CDN	Geographical	Oracle	CDN	MS-Stream	16 servers
Random	Geographical	Oracle	Random	MS-Stream	16 servers
RR	Geographical	Oracle	Round robin	MS-Stream	16 servers

4.4.3 Quality of Experience

To evaluate the end-users QoE, three main metrics are considered: the number of rebuffering events on Figure 4.7, the average video bitrate displayed on the user video player (Figure 4.8) and the number of quality changes during the session (Figure 4.9).

`Muslin` clients were able to reach a higher QoE compared to most current setups, as we demonstrate an increase of 100 kbps in median displayed bitrate, 2.5 less quality changes per minute, and almost no rebufferings compared to a best-case *CDN* implementation. The bitrate increase is due to the dynamic provisioning of content servers based on the

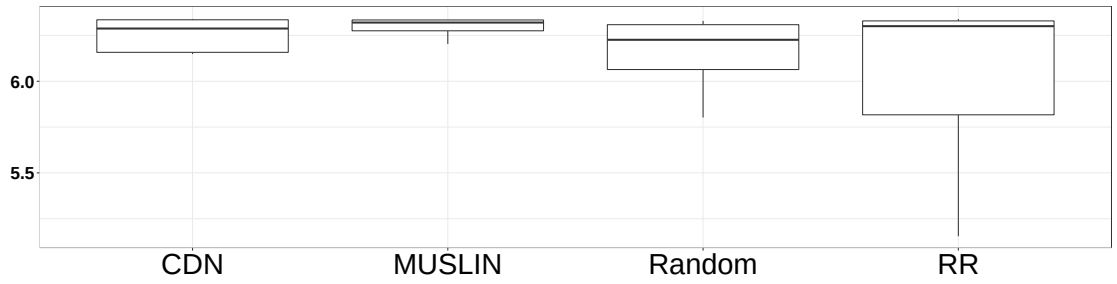


Figure 4.8: Displayed bitrate (Mbps), selected setups

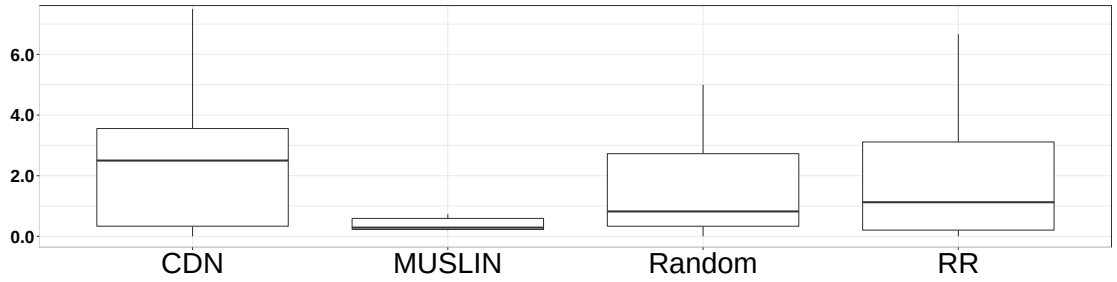


Figure 4.9: Quality changes per minute, selected setups

actual clients demand. The quality changes and rebufferings decreases are a consequence of RS_{sc}-based servers selection, which prioritizes servers with available bandwidth and high response rates.

4.4.4 QoE fairness

In this subsection, QoE fairness between clients is discussed. As shown above, `Muslin` median QoE results are better than a best-case *CDN* implementation, and the distributions are less spread than other setups, as the fairness among users is higher.

Table 4.6: QoE fairness (F index), selected setups

QoE metric	CDN	`Muslin`	Random	RR
Bitrate	0.7727	0.9610	0.5952	0.4685
Quality changes	0.4551	0.9485	0.5408	0.4660
Rebufferings	0.6952	0.9095	0.5179	0.6452

Indeed, as shown in Table 4.6, we registered an increase of 19.6% in bitrate fairness, 52% in quality changes fairness and 23.6% in rebufferings fairness, using the F index (based on standard deviation, see Equation 4.7) described by T. Hoßfeld *et al.* [HSKHV16]:

$$F = 1 - \frac{2\sigma}{H - L} \tag{4.7}$$

The main reason for such increases is the feedback-based RS_{sc} computation, enabling to advertise the most suitable servers for each client, which are not necessarily the closest ones. It also spreads the load evenly across all servers, and avoids starvation that may happen for some clients in a traditional CDN scheme.

4.4.5 Network overhead

As stated in Section 2.2.3, MS-Stream can use some redundancy in sub-segments to reduce the number of rebufferings in case of server or network impairment. Figure 4.10 shows the total network overhead percentage required by MS-Stream clients for a few selected setups.

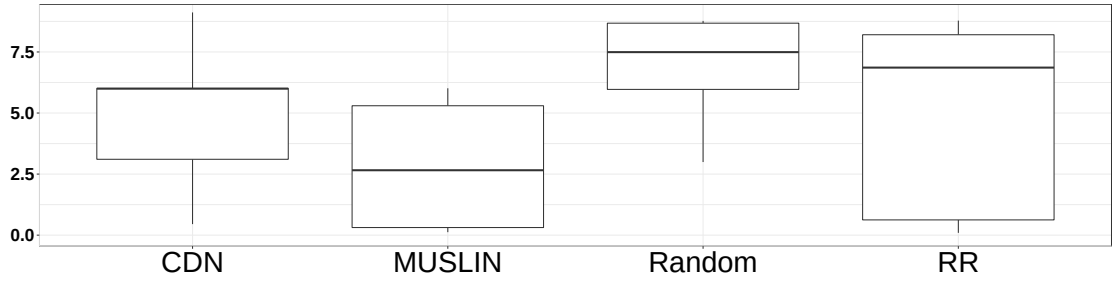


Figure 4.10: Network overhead (%), selected setups

MS-Stream manages to lower the required network overhead, as `Muslin` dynamically provisions servers and advertises more suitable content servers to clients. Indeed, the MS-Stream clients detects that most servers are able to reply in time to video segments requests, and thus lowers the redundancy in sub-segments requests. On the contrary, when servers are selected randomly, the network overhead required is higher as the delivery of sub-segments is inconsistent.

4.4.6 Experiments summary and discussion

`Muslin` manages to increase QoE and fairness while lowering provisioning costs by combining dynamic provisioning with feedback-based servers selection and multiple-source content delivery. QoE and fairness increases compared to a best-case CDN setup

are due to the servers selection taking server load and bandwidth into account and not only distance (thanks to the RS_{sc} ranking score). In our experiments, West coast and California CDN servers are particularly stressed as they are close to large clients pools. In a CDN setup, even if the audience is correctly estimated prior to the streaming session, all clients will contact the nearest server and might top off the maximum capacity of particular CDN servers in specific zones, thus reducing QoE and fairness.

4.5 Conclusion

Current video streaming solutions usually rely on CDN servers and HAS techniques to deliver content. However, the client is often bound to a single server, thus prone to failures, congestion or unavailability, and therefore unfairness between users. Besides, providers who statically over-provision their platform to mitigate these issues face a higher cost. To overcome these challenges, we presented our first contribution, `Muslin`, a multi-source live streaming system which manages to reach higher QoE and fairness than currently adopted streaming systems. `Muslin` takes into account clients' real-time feedbacks, dynamically replicates content and improves server advertising to enhance users' QoE and fairness, while minimizing the required infrastructure scale (*i.e.*, cost). We showed in our experiments that thanks to the coupling of MS-Stream with the proposed `Muslin` system, end-users experienced almost no rebufferings, a higher video bitrate, and more evenly shared QoE, compared to existing state-of-the-art streaming systems setups.

To further reduce costs, an alternative to CDN servers is P2P and edge-assisted streaming, which provides better scalability by essence. However, traditional P2P streaming is unstable, unreliable, and lacks privacy preservation, as peers and providers can access sensitive data. Our next step is thus to further improve scalability while providing strong privacy guarantees and unaltered QoE to end-users, thanks to edge-assisted multiple-source streaming.

Summary

`Muslin` uses dynamic server provisioning and advertising based on real-time delivery conditions combined with MS-Stream to provide a better QoE at the lowest cost. Compared to a best-case *CDN* setup, `Muslin` provides:

- +100 kbps video bitrate / +19.6% fairness
- -2.5 quality changes per minute / +52% fairness
- 0 rebufferings / +23.6% fairness
- -18% cost (as server time)

Chapter 5

PrivaTube: Privacy-preserving edge-assisted streaming

Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say.

— Edward Snowden

When using online video streaming services such as `Muslin`, each user generates a history of watched videos. The platform provider can use this data for personalized recommendations for new videos, or for targeted advertising. This can lead to major threats to privacy as it is possible to infer private information about the user, such as his gender, his origin, and his political, religious or sexual orientation.

In this chapter, we present PrivaTube [DSBMC+19], a *practical* and *privacy-preserving* video streaming system.

5.1 Introduction

Very few privacy-preserving video streaming systems exist. Most of them rely on heavy cryptographic mechanisms over multiple nodes, adding a performance and latency overhead, effectively reducing QoE. Unlinkability-based techniques (*i.e.*, fake requests) are efficient but come at the cost of bandwidth and storage overheads. It is therefore challenging to provide both privacy guarantees and high QoE.

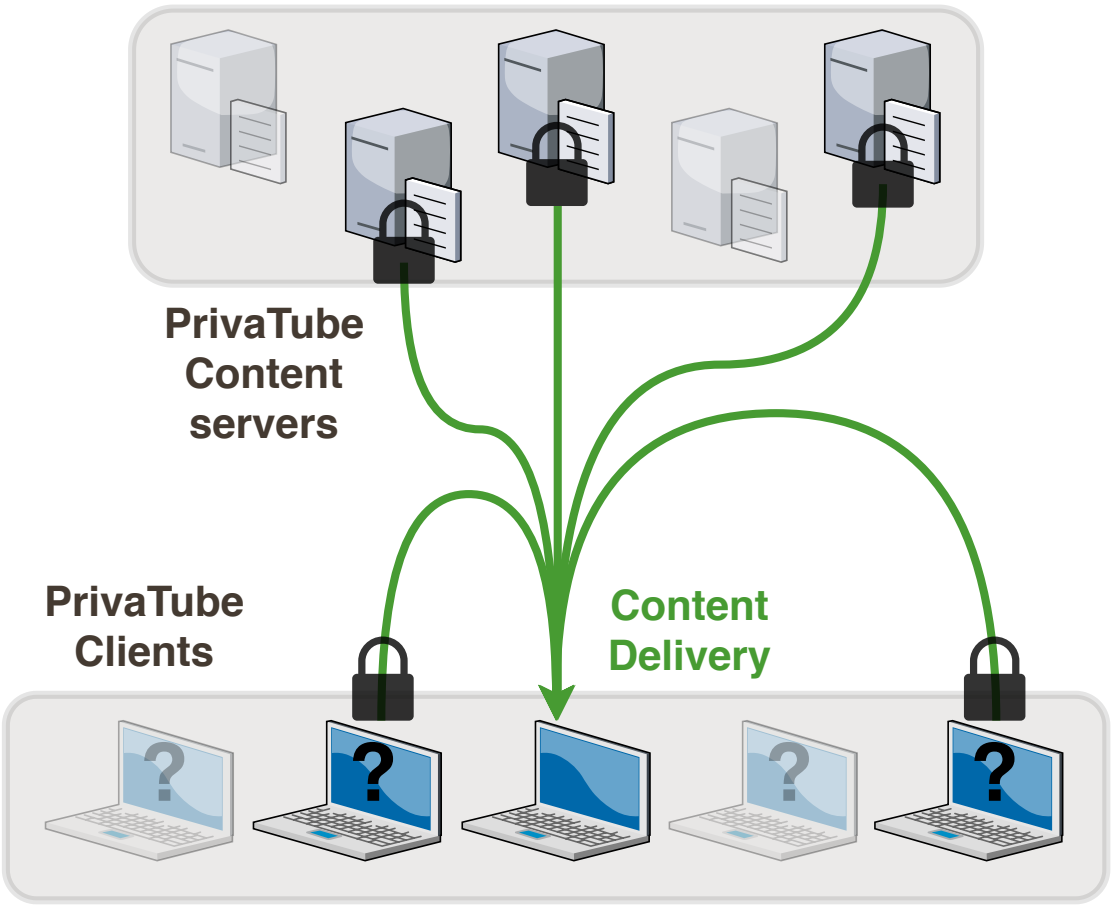


Figure 5.1: PrivaTube illustration

PrivaTube aims to provide strong privacy guarantees with unaltered QoE and to reduce costs even further than `Muslin`. PrivaTube is able 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 (see Figure 5.1). Indeed, PrivaTube extends MS-Stream [BQLN+18] (see Section 2.2.3), a protocol for video streaming using multiple sources, compatible with the leading MPEG-DASH standard [Sod11]. It ensures that the load on core servers is minimized, and that the impact of timeouts and network failures 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 TEEs at both the client and server sides.

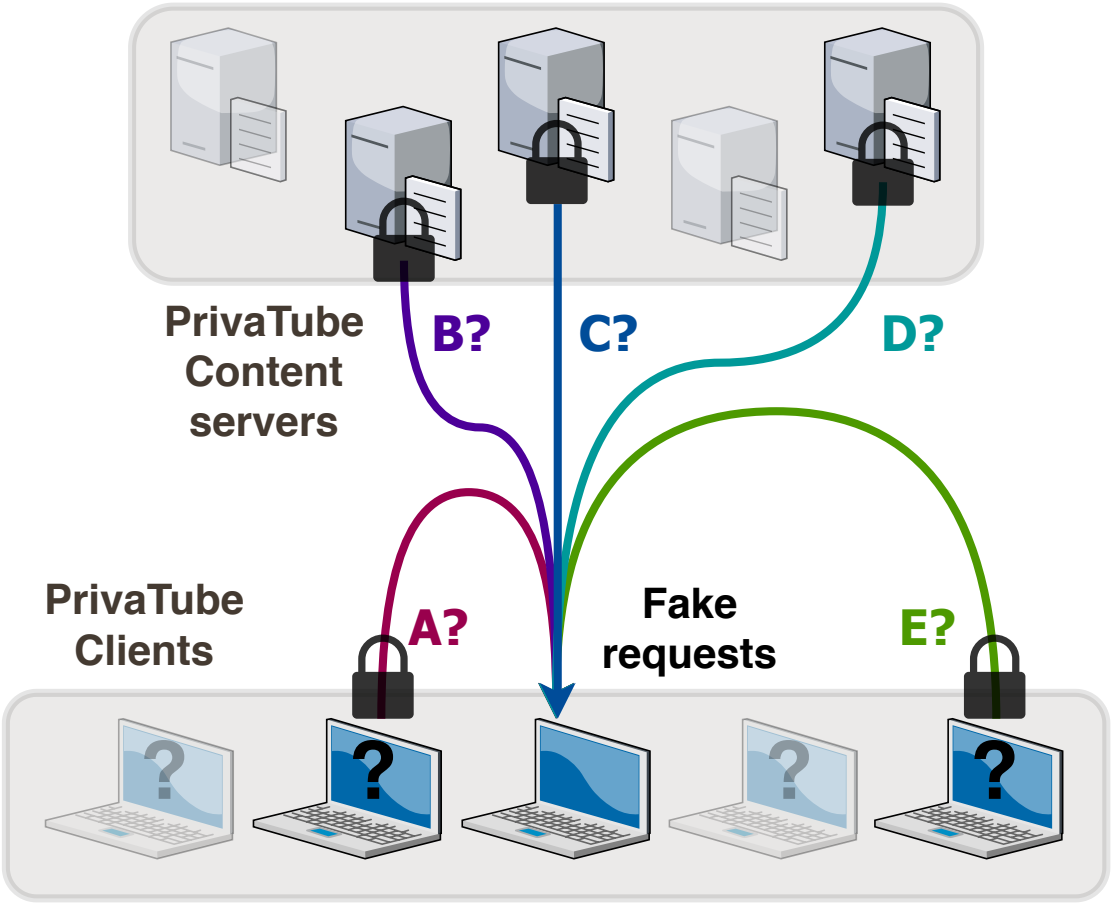


Figure 5.2: PRIVATUBE fake requests illustration

PRIVATUBE further prevents assisting peers from inferring histories based on assistance requests by introducing fake requests (see Figure 5.2). 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. Besides, PRIVATUBE

outperforms all current privacy-preserving video streaming systems by up to several orders of magnitude in terms of latency and video bitrate (*e.g.*, quality), as it has a negligible impact on performance.

This chapter is organized as follows. We present an overview of the constituents and privacy objectives of PrivaTube in Section 5.2. We detail how the system scales and provides high QoE in Section 5.3, and how it preserves privacy in Section 5.4. We discuss our contributions and provide a security analysis in Section 5.5. We implement PrivaTube and deploy it on 14 SGX-enabled machines, then perform an extensive evaluation, including micro-benchmarks, macro-benchmarks and a large-scale simulation, all presented in Section 5.6. Finally, we conclude in Section 5.7.

5.2 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.

Service model

We target the 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 QoE in terms of (1) bitrate, (2) quality fluctuations, (3) video interruptions and (4) startup time, as they are the main factors impacting QoE [SES+14].

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 [Peeb]. 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 5.3, while privacy preservation is the focus of Section 5.4.

5.3 Practical and High-QoE Streaming

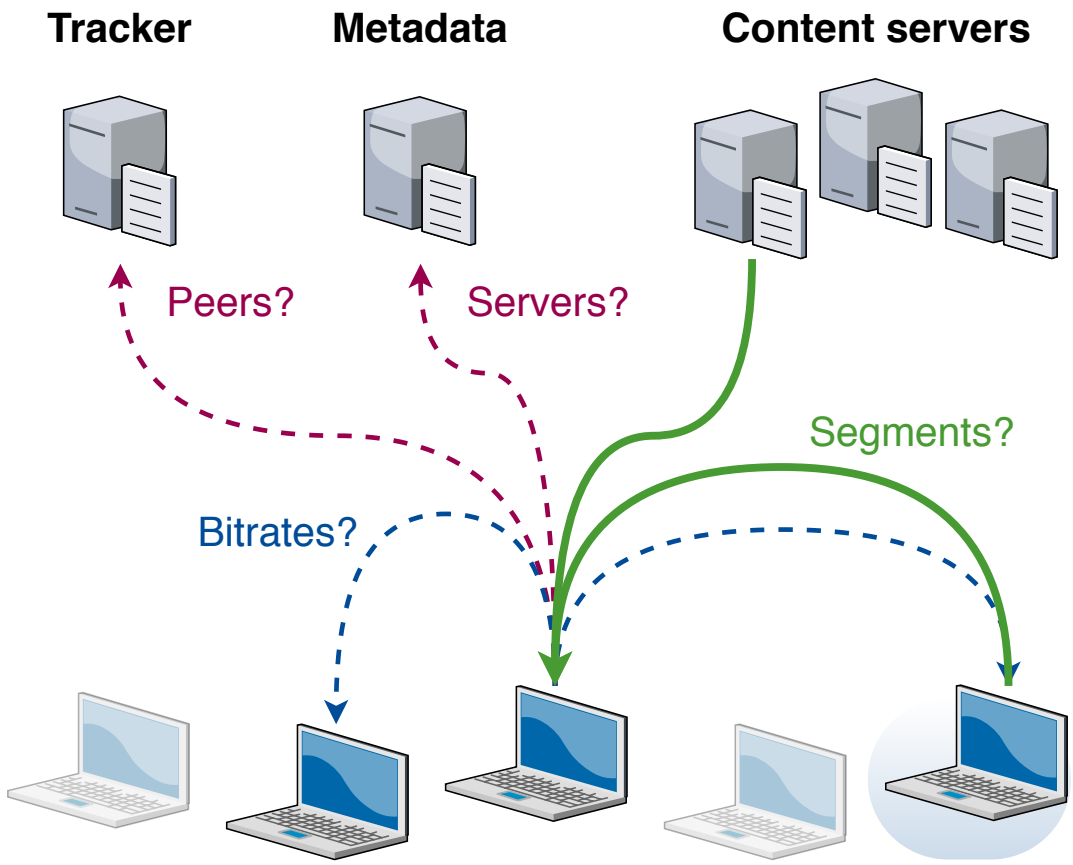


Figure 5.3: Streaming using video servers and assisting peers

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

5.3.1 Edge-assisted Content Delivery Network

To address the limited capacity of dedicated servers in providing video content to users, we leverage an edge-assisted CDN. Figure 5.3 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.

5.3.2 Adaptive Streaming

The PrivaTube streaming protocol extends the MS-Stream solution [BQLN+18] (see Section 2.2.3) and is fully compatible with DASH. Figure 5.3 presents the complete workflow of our extension.

Selection of assisting peers and servers

The selection of servers and sub-segments in the previous version of MS-Stream exclusively favors QoE for the client, but does not consider different classes of servers [BQLN+18]. 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[1] keeps track of the video access history of clients. It returns to the client a random subset of up to 50 CAP (② in Figure 5.3). 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 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 5.3).

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

[1]We consider a single server for the tracker and for the metadata server in our implementation. They are both built as stateless service tiers over NoSQL databases, offering excellent horizontal scalability through already-available solutions [KPW+19]. We keep this extension for future work.

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 5.3).

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.

5.3.3 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.

5.4 Privacy

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. 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 5.3, 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.

5.4.1 Trusted execution environments

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

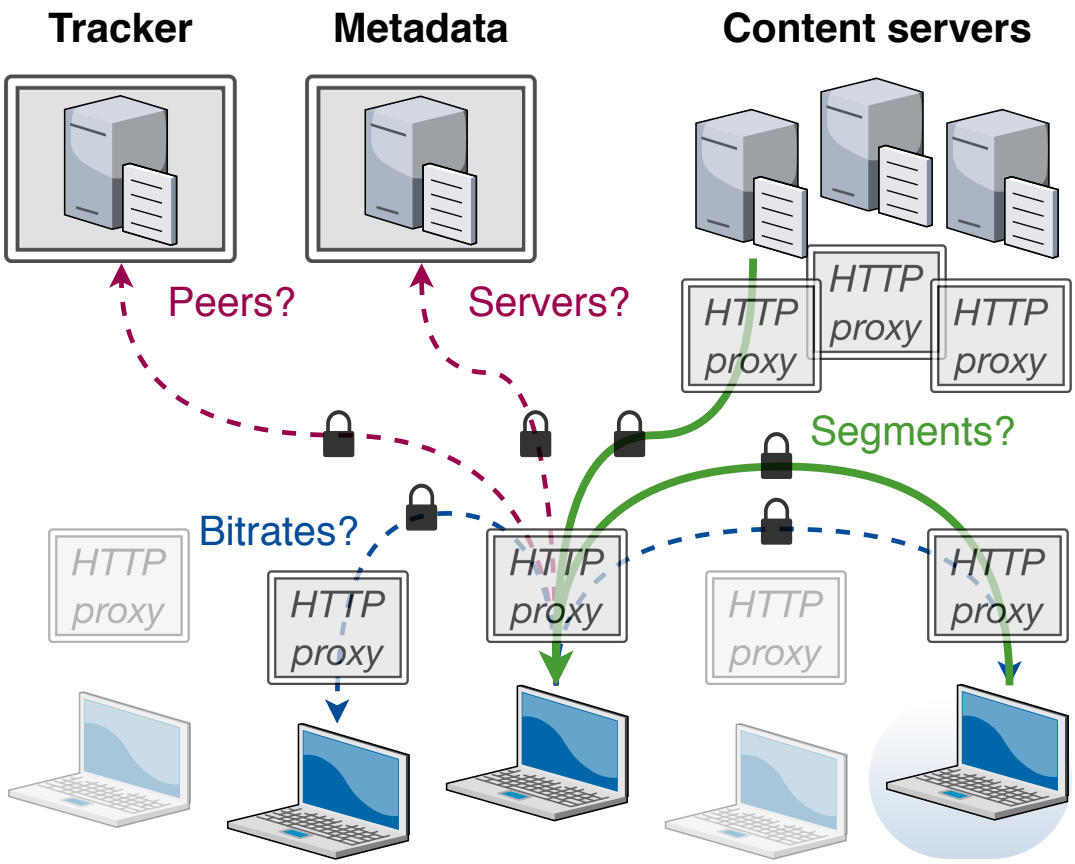


Figure 5.4: PRIVATUBE architecture for privacy preservation through HTTP proxies and servers inside SGX enclaves

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 5.4. 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 [GJP12].[2] 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.

5.4.2 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,

[2]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.

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.

5.5 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.

5.5.1 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 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:

$$\mathrm{TFR}_t = (\mathrm{TLR}_t \times \frac{1-\delta}{\delta}) - \sum_{i=1}^{f} \mathrm{FR}_t(i) \tag{5.1}$$

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.

5.5.2 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 [ARM]) 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 Priva-

TUBE, 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 chapter. 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 [LCW+06, DMPQ16, GHK+10, BCE+07, VRS+18]. 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 on its own. 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 [MM09] or on the use of cryptographic primitives [GKP+17]. Nevertheless, as these solutions cannot provide recommendations with unaltered performance and accuracy, we designed `PProx`, a privacy-preserving recommender system (see Chapter 6).

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 [VBMW+18, LSG+17]. 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 [OTK+18] 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 [CZRZ17]). This

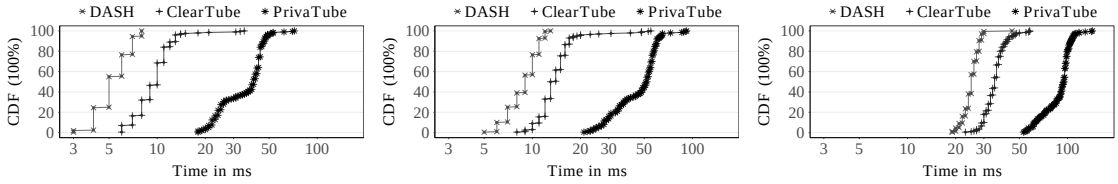
requires periodically rotating the long-term secrets provisioned to PrivaTube client enclaves. Each rotation could spawn a Diffie-Hellman key agreement session, similarly to the Intel SGX attestation procedure. This mechanism would achieve *forward secrecy*: compromising a long term enclave private key does not compromise the session keys.

5.6 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? (Section 5.6.2)
- Is the use of assisting peers successful in improving QoE for the clients, and does it help to reduce the load on video servers? (Section 5.6.3)
- Are fake requests effective in hiding clients' access patterns to videos, and in improving performance and usefulness of assisting peers? (Section 5.6.4)



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

Figure 5.5: 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.

5.6.1 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

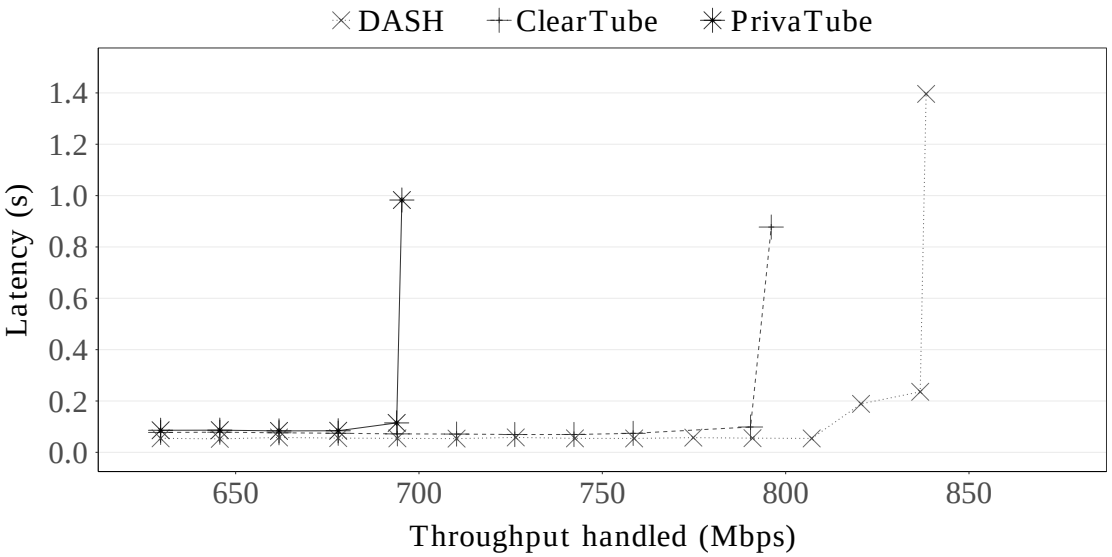


Figure 5.6: Throughput and latency for DASH, ClearTube and PrivaTube, without using network emulation. The inflection shows the saturation point of the three solutions.

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) [DAS]. 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.[3] The SD and HD are the two available high-quality (HQ) variants. As detailed in Section 5.3, 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

[3]The exact size depends on the codec and nature of the video.

video servers and assisting peers) and fetch a new segment as soon as one buffer slot is consumed.

We compare PrivaTube to two baselines.[4] The first baseline is the DASH standard. Our DASH implementation uses the Nginx high-performance HTTP server [NGI]. The second baseline is PrivaTube without enabling any of the mechanisms for protecting users' privacy. This corresponds to the system as described in Section 5.3, without any of the additions detailed in Section 5.4. 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.

5.6.2 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.

Our focus is on the throughput and latency limitation of one video server. As a result, 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]

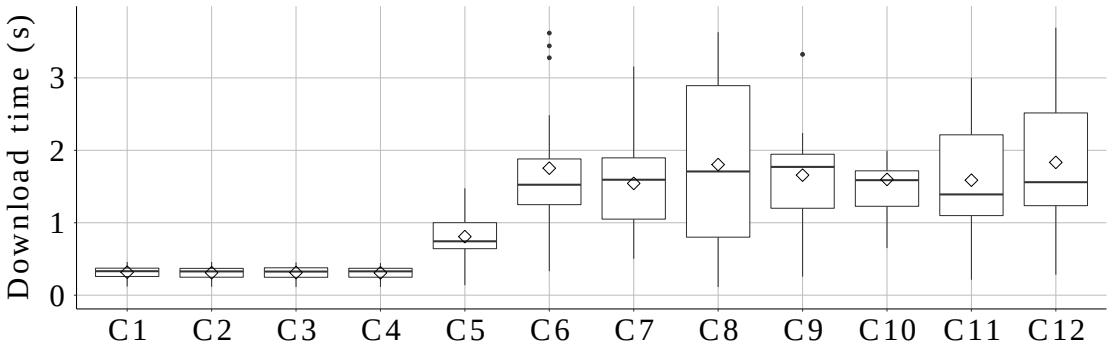
We first focus on request latencies. We set up a single client performing 200 consecutive requests for a 6-second video segment. Figure 5.5 presents the cumulative distributions of retrieval delays for the three available bitrates, using the three systems. Unsurprisingly, DASH provides the lowest latencies, and latencies increase with the segment size. The

[4]We choose not to evaluate PrivaTube against indirection-based solutions such as Tor [DMS04] as we aim at providing a high QoE, incompatible with the latency and bandwidth Tor provides. Besides, a direct comparison with edge-assisted solutions such as Popcorn [GCM+16], Xunlei Kankan [ZLHC14] or LiveSky [YLZ+09] is difficult, as there is no publicly-available implementation of these systems.

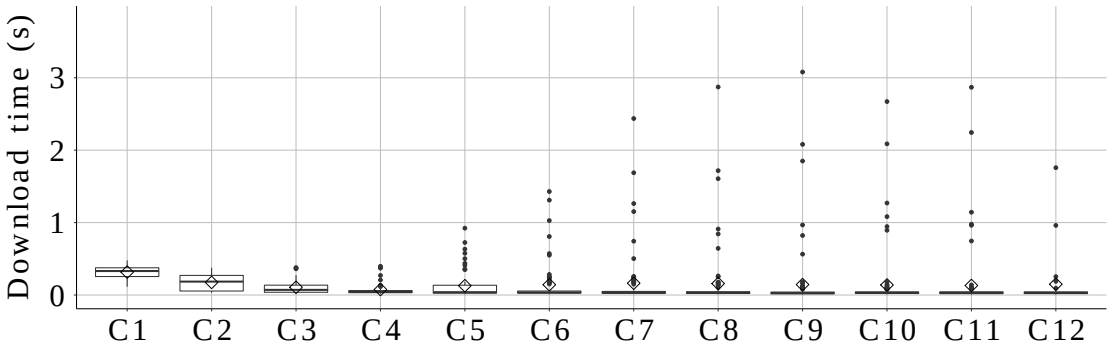
[5]One could cache pre-computed segments at a PrivaTube video server, but we did not implement this optimization.

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 versus 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 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 versus 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.

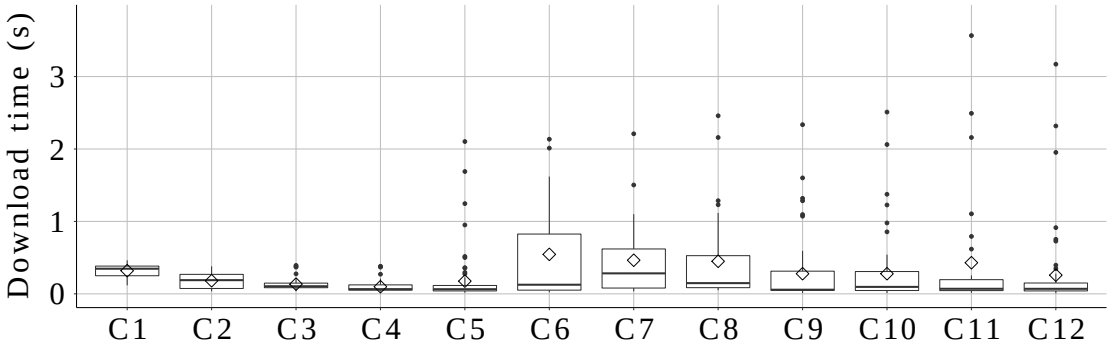
We now focus on achievable throughput for a video server under a concurrent request workload. We use only the HD quality, with video segments of size 2 MB. We set up a client with the `wrk2` workload injection tool for HTTP requests [Ten18]. We use 4 injection threads from a single client, after checking that this setup is enough to saturate all three configurations. Figure 5.6 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 5.5c). 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.



(a) DASH



(b) ClearTube

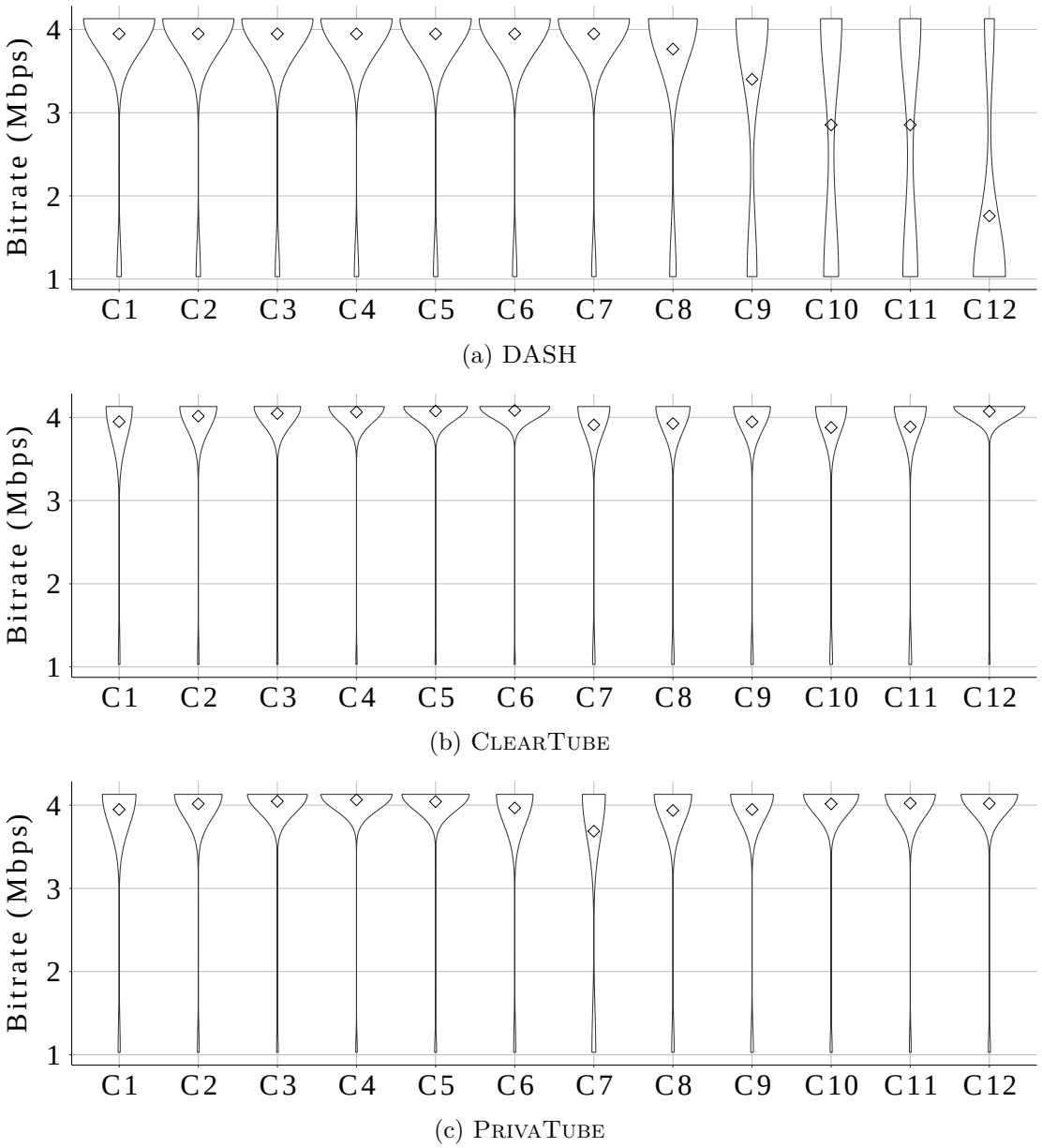


(c) PrivaTube

Figure 5.7: Distributions of segments download times

5.6.3 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.



(a) DASH

(b) ClearTube

(c) PrivaTube

Figure 5.8: Distribution of achieved playback bitrates

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 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 solutions differ greatly in achieved quality rate and its stability, directly impacting QoE.

Figure 5.7 presents the distribution of segment download times, while Figure 5.8 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 5.7b and 5.7c). 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 5.7b 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 5.8). 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.

5.6.4 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 5.5.1. 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 [HK15a] 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 5.9. It is a heavy-tail distribution typical of VoD systems [CPK95, YZZZ06]. 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.

[6] We chose 2014 as it is the last available full year from MovieLens 20M dataset (01/1995 to 03/2015).

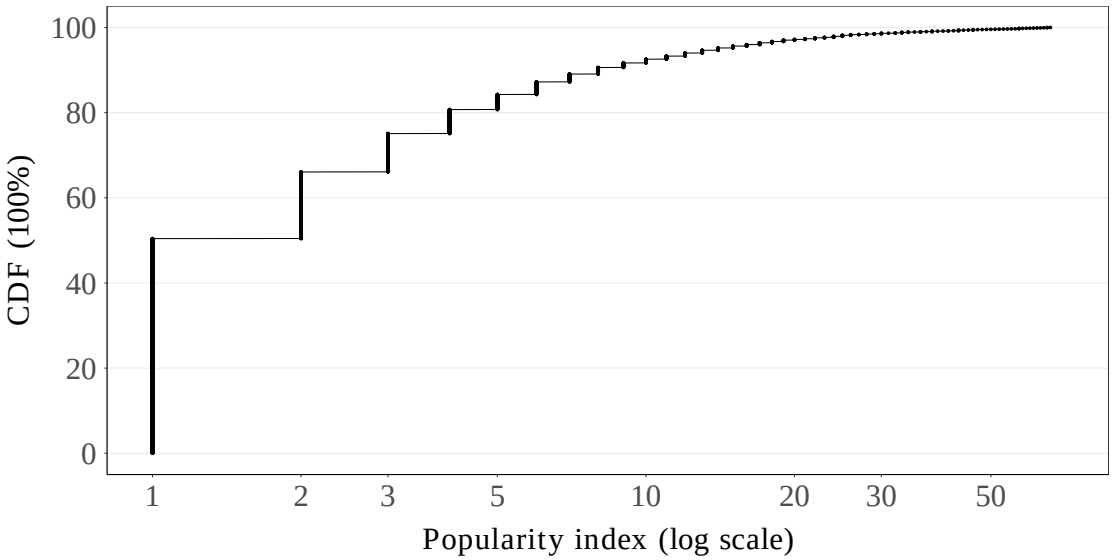


Figure 5.9: Distribution of movies popularities in MovieLens

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 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 5.4 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

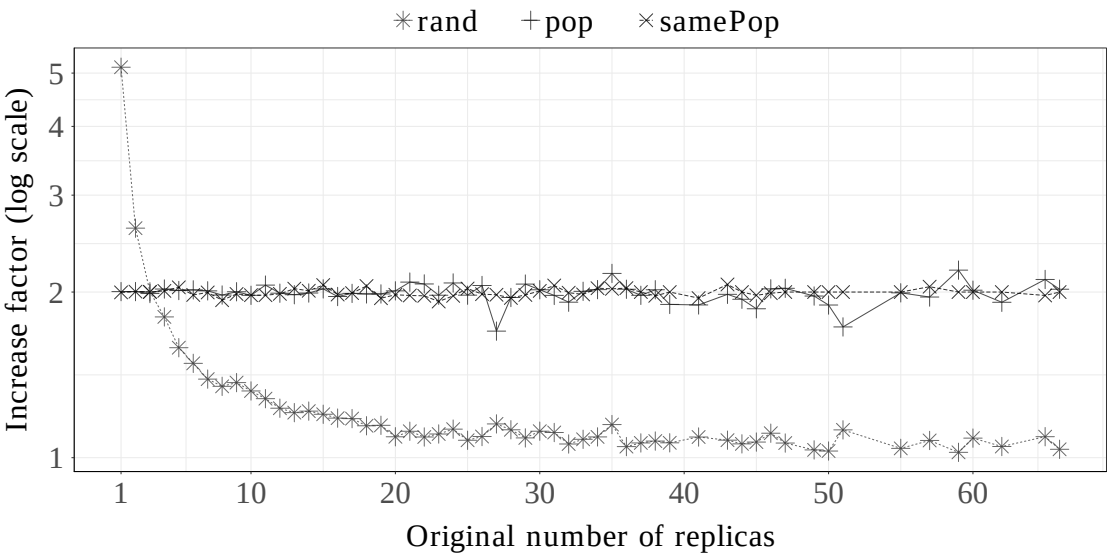


Figure 5.10: Replicas increase factor, $\delta = 50\%$

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 $v_{1^{\text{st}}}$. 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 $v_{2^{\text{nd}}}$. We define the *replica increase factor* as the ratio $v_{2^{\text{nd}}}/v_{1^{\text{st}}}$. Figure 5.10 presents the distribution of this metric, as a function of $v_{1^{\text{st}}}$, when exactly one fake request is sent for each video access ($\delta = 50\%$).

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 5.1. 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.

Table 5.1: Replicas increase factor for various values of δ

# fake req.	$\delta = 66\%$ $\times$ 0.5			$\delta = 50\%$ $\times$ 1			$\delta = 25\%$ $\times$ 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

5.7 Conclusion

Access histories can reveal critical personal information, and centralized video streaming solutions are notorious for exploiting personal data. The platform provider can use this data for personalized recommendations for new videos, or for targeted advertising. This can lead to major threats to privacy as it is possible to infer private information about the user, such as his gender, his origin, and his political, religious or sexual orientation. Hiding the interests of users from servers and edge-assisting devices is necessary for a new generation of privacy-preserving streaming services. However, very few privacy-preserving video streaming systems exist. Most of them either rely on heavy cryptographic mechanisms, adding a performance and latency overhead, effectively reducing QoE, or on unlinkability-based techniques (*i.e.*, fake requests), with high bandwidth and storage overheads. Therefore, the state of the art was still lacking an efficient streaming system providing both strong privacy guarantees and unaltered QoE.

To tackle this challenge, we presented our second contribution, 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. Indeed, the provided QoE is far greater than traditional HAS streaming, with up to +300% video bitrate (*i.e.*, quality). The main

downside is a 36ms to 71ms longer startup delay due to encryption and proxying, which is negligible in practice.

To further improve privacy while providing a high QoE, one must implement a privacy-preserving recommender system to complement PrivaTube. Indeed, video streaming platforms usually provide recommendation services to retain users on their website or application. Yet, they pose a serious threat to privacy, as user profiles are established based on the watching history. Our next step towards an unaltered QoE is thus to provide privacy-preserving recommendations to end-users without compromising on quality and performance.

Summary

PrivaTube extends MS-Stream to offer a high QoE by aggregating video content from multiple servers and edge peers. Users' privacy is preserved through encryption in HTTP proxies running in Intel SGX enclaves, and fake requests to obfuscate access patterns. Fake requests are further leveraged to implement proactive provisioning and improve QoE. Compared to a typical DASH setup:

- 2× to 15× faster video segments download
- +40ms slower startup delay (first segment)
- +10% to +300% video bitrate
- -17% raw server throughput

Chapter 6

PProx: High-QoE privacy-preserving Recommendation as a Service

The fantastic advances in the field of electronic communication constitute a greater danger to the privacy of the individual.

— Earl Warren

Video streaming platforms (such as YouTube [You20], Vimeo [Vim20] or Dailymotion [Dai20]) often feature recommendations for similar content to end-users. They rely on these recommendations to retain users on their website or application. To do so, they establish a user profile based on the watching history. This leads to threats to privacy as (i) service providers gather private data on each user, (ii) an adversary can intercept recommendations and deduce private information about the user, or (iii) malicious platform providers can display targeted ads tailored to a specific user instead of a generic profile to generate income.

To this end, we present PProx, an efficient and easily-deployable solution for privacy preservation in recommendation engines and RaaS.

6.1 Introduction

A few privacy-preserving recommender systems exist. They can be either cryptography-based, usually very slow (several seconds latency); differentially private with added noise,

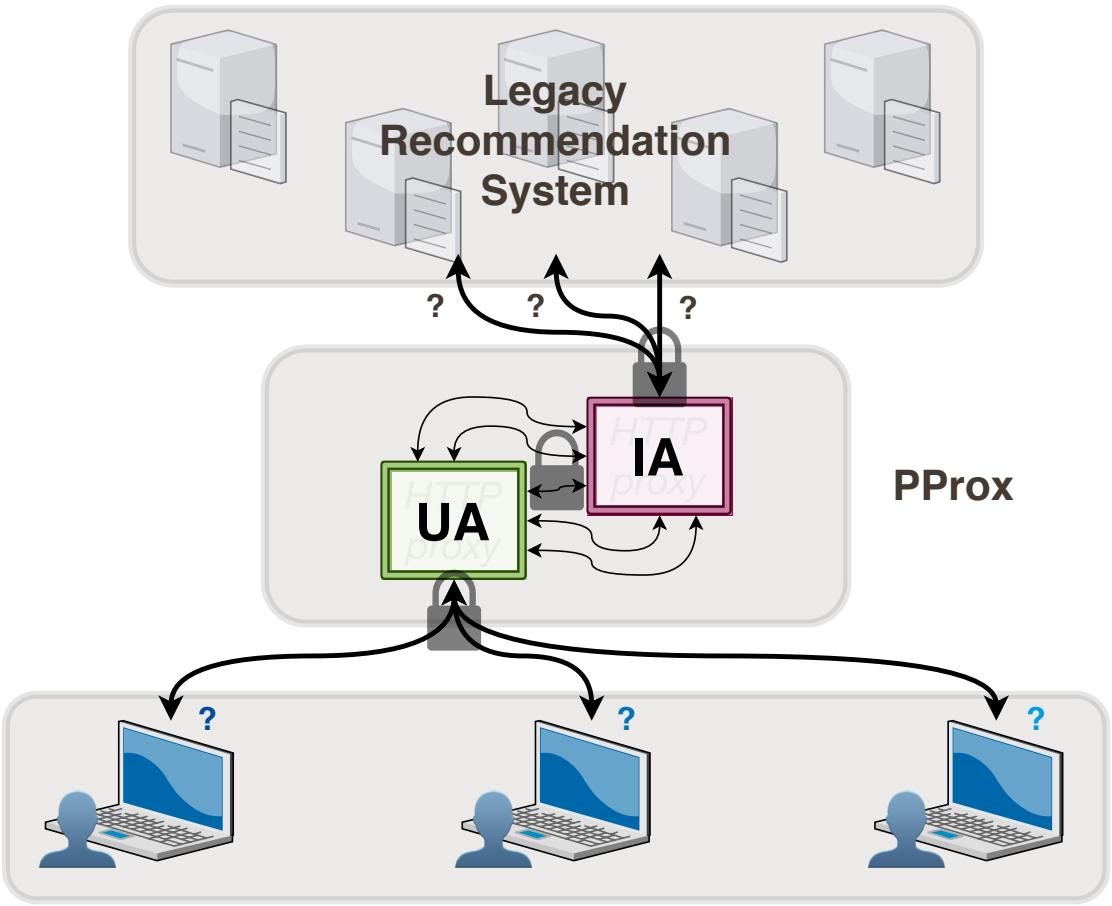


Figure 6.1: PProx illustration

thus inaccurate; or P2P-based and decentralized, often slow, less accurate and unreliable. Besides, all of them use specific recommendation algorithms, and require to install a heavy code layer at the client side. Current reliable and accurate privacy-preserving recommendation systems have a latency between 10 and 100 seconds on a high-end server CPU. It is therefore challenging to provide privacy-preserving recommendations without compromising on quality and performance.

To tackle these issues, PProx introduces a privacy-preserving proxy service, standing between users and any off-the-shelf unmodified Legacy Recommendation System (LRS). This proxy service intercepts feedback insertions and requests for recommendations. It pseudonymizes on the fly the user and items identifiers, and hides links between the two. This guarantees unlinkability between clients and the items they access or receive as recommendations. The deployment of PProx does not require to provision private keys or

models to the user side. PProx does not modify in any way the requests results returned by the LRS (*e.g.*, by adding noise or returning an overset of results) and its use is totally transparent for the users.

PProx leverages the support in modern cloud infrastructure for a Trusted Execution Environment (TEE), specifically Intel SGX [CD16b], allowing to run secure enclaves on untrusted hardware. In contrast with earlier work using SGX to protect the privacy of access to online services [KCG17, KPW+19, MBF+17b] the design of PProx acknowledges the possible vulnerability of Intel SGX to side-channel attacks [WCP+17, GESM17, BMD+17, CCX+19, SLM+19, VBMW+18, MIE17]. In addition, it considers the vulnerability of the cloud infrastructure to timing attacks on network flows resulting from clients' interactions with the recommendation engine.

To prevent an adversary from breaking unlinkability properties using a side-channel attack, PProx implements a data partitioning principle where the information necessary to link a user to a specific item or recommendation is split between two layers running in different SGX enclaves. This is based on the observation that implementing side-channel attacks on multiple nodes in a limited time is unlikely to happen synchronously, leaving time to detect breaches [CZRZ17, GLS+17, OTK+18] and take appropriate countermeasures. Protection against timing attacks is achieved with the help of request and response shuffling, hiding the correlation between flows while respecting tight bounds on additional service latency (see Figure 6.1).

PProx is integrated with Harness [Acta], an open-source machine learning platform, and its *Universal Recommender* [Actb] module. Harness is representative of a LRS used by a RaaS provider: it supports high-throughput and low-delay operations, and scales horizontally to serve growing user bases. PProx is also able to similarly scale horizontally to handle varying load while minimizing the impact of privacy-preservation on performance.

Our evaluation over a 27-node/54-core Kubernetes cluster of Intel SGX-capable NUC servers, and using a real-world workload, shows that PProx is able to efficiently protect privacy while respecting strict end-to-end latency objectives, and to scale up to handle increasing workloads in unison with the scaling of the LRS. A single instance of PProx can handle 250 requests per second using 4 cores, and it scales up to 1.000 requests per second using 4 proxy instances, matching the capacity of a 32-core deployment of Harness.

We detail our system and adversary model in Section 6.2. We give a high-level overview of PProx in Section 6.3. We present the construction of the proxy service in

details in Section 6.4, and its implementation in Section 6.7. We discuss the security of PProx in Section 6.5. We overview the integration of PProx with Harness universal recommendation engine in Section 6.6. We evaluate the resulting system on a 27-node Kubernetes cluster and present our results in Section 6.8, then conclude in Section 6.9.

6.2 System model and objectives

We start by defining our system model, our assumptions, our security objectives, and the power of the adversary.

6.2.1 System model

Figure 6.2 illustrates the constituents of the system. Users interact with a website or application offering access to items, *e.g.*, books, news articles, or movies (①). This service outsources the management of a recommendation feature embedded in its front-end to a Recommendation-as-a-Service solution running in a public cloud (②).

The RaaS runs a Legacy Recommendation System for the application (③), accessed via a simple REST API. Call $\texttt{post}(u, i[, p])$ allows user u to send feedback to the recommendation engine about access to item i with an optional payload p, if required by the recommendation algorithm. For instance, a movies recommender may leverage ratings by the user, while a recommender for items in an online store may only require identifiers. Call $\texttt{get}(u)$ returns a collection of n items $(i_1, \ldots, i_n)$ recommended to user u.

PProx introduces an additional component, the privacy-preserving proxy service (④), lying between the clients and the LRS. It runs as part of the RaaS in the same public cloud as the LRS to avoid indirections through multiple data centers and the resulting impact on latency.

A thin user-side library, easily embeddable in the application or web front-end as static web code, and offering the exact same REST API as the LRS, intercepts, encrypts and forwards clients' API calls to the proxy service, and, in the case of get calls, returns the list of recommendations (⑤).

6.2.2 Trust and operational assumptions

The user-side library is considered trusted for the processing and handling of personal data, *i.e.*, its code is public, in an interpreted language, and it can be audited by external

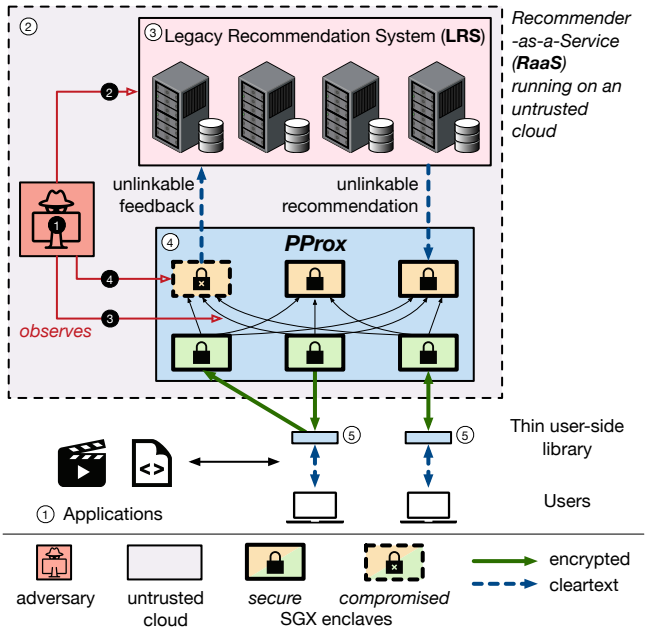


Figure 6.2: PProx system constituents (①-⑤ in §6.2.1) and adversary model (❶-❹ in §6.2.3)

parties. We also trust the user: protecting against a compromised browser or application is orthogonal to this work.

The LRS and proxy service run on an untrusted public cloud, subject to attacks and possible data leaks. We do not wish, therefore, to trust this infrastructure for processing information in the clear. We assume, however, the availability in this public cloud of a Trusted Execution Environment. Our implementation uses specifically Intel SGX [CD16b]. We trust Intel for the certification of genuine SGX-enabled CPUs, and we assume that the code running inside enclaves is properly attested before being provided with secrets.

A LRS is typically built over data processing frameworks, *e.g.*, Apache Spark, and databases, *e.g.*, MongoDB, preventing from using source-based application partitioning techniques such as Glamdring [LPM+17]. As the data that the LRS uses is almost entirely of sensitive nature, the trusted computing base is potentially very large, preventing the use of full-application containment, *e.g.*, using SCONE [ATG+16]. It is not desirable, under these conditions, to run the LRS itself inside SGX enclaves and we reserve their use for the proxy service, which we design to take into account TEE constraints.

6.2.3 Privacy objectives and adversary model

The goal of PProx is to preserve *User-Interest unlinkability*. It should be impossible for an adversary to relate a specific user (as identified by their identifier or any unique characteristic, *e.g.*, their IP address or geographical location) to an access to an item or of their possible interests as reflected by received recommendations. More formally, it should be impossible for an adversary to (1) learn that a user u called $\texttt{post}(u, i[, p])$ for item i and (2) that a user u received a recommendation for an item i following a $\texttt{get}(u)$ call.

We consider a powerful adversary (Figure 6.2, ❶). This adversary wishes to break the unlinkability property by observing all components of the RaaS backend. It does not, however, interfere with the functionality of the system, as it does not attempt to manipulate or bias the recommendations returned by the LRS, and it does not block or delay the access to the service for specific users or specific applications.

As the cloud is untrusted, we consider that the adversary can successfully attempt to break security measures put in place by RaaS providers, such as system-level access control or the use of secure connections (TLS/SSL) to and from its clients [WXW14]. We consider, therefore, that the adversary may see all API calls to the LRS in the clear, and can access any data manipulated by the LRS when computing recommendations (❷). Similarly, the adversary may have control over the public cloud infrastructure including its network appliances and we assume it is able to observe network connections both from outside and within the data center (❸).

Our design takes into account the possibility, highlighted by recent work, of time-based or cache-based side-channel attacks on SGX [WCP+17, GESM17, BMD+17, CCX+19, SLM+19, VBMW+18, MIE17], allowing an adversary to access the secrets that were provisioned to an enclave. This contrasts with previous designs that consider enclaves as inviolable [ATG+16, KPW+19, KCG17, MBF+17b]. Mechanisms such as Cloak [GLS+17], Déjà Vu [CZRZ17] or Varys [OTK+18] allow, on the other hand, to detect the occurrence of such attacks and to respond appropriately (*e.g.*, by shutting down the system and restart it after a security audit and using new secrets). Our model includes, therefore, the possibility for the adversary to compromise and break into a single enclave at a time, on any server. For instance, we illustrate in Figure 6.2 that the top-left enclave of the privacy-preserving proxy service has been compromised and its secrets leaked to the adversary (❹).

6.3 PProx in a nutshell

In addition to its privacy objectives, PProx targets the ability to sustain the load achievable by the LRS, the compliance with RaaS service-level objectives, and ease of deployment. First, it must scale to large RaaS installations, with a potentially high throughput of user requests for both insertions of feedback and collections of recommendations. We assume that the LRS system scales horizontally (*i.e.*, by adding more machines). The privacy-preserving proxy service must, therefore, also scale horizontally. At the same time, the amount of resources required for enabling privacy, and the additional costs that will have to be supported by the application (as a client of the RaaS), must remain within a fraction of the costs required to operate the LRS itself. Second, the interaction of users with the RaaS solution must happen with small delays (typically, at most a few hundred milliseconds for the overall processing of the request in addition to the network delay to/from the user)[1]. Finally, ease of deployment requires that the integration with the website or application only relies on static code and globally known information, does not require the intervention of users and does not require maintaining state specific to a RaaS service across sessions.

A two-layer privacy-preserving proxy service. At the core of PProx is a proxy service that guarantees that (1) the LRS only sees pseudonymous information, for both user identifiers and item identifiers[2] and (2) that it is impossible to relate a call from some user to a call sent to the LRS (and similarly for responses from the LRS to the user).

We start by observing that mapping a user identifier to a pseudonym in a single SGX enclave acting as a proxy and forwarding the pseudonymized request to the LRS is not sufficient under our adversary model. The adversary may, indeed, compromise this single enclave and learn the direct associations between user identifiers and item identifiers. PProx uses instead a two-layer proxy service, with the two layers running in distinct SGX enclaves on different servers. The foundational principle of this design is that no enclave is provisioned with all the secrets necessary to an adversary to break unlinkability:

[1]It is not desirable, for these reasons, to rely on external anonymity services such as AnonyFlow [MSO12] or Tor [DMS04], for their lack of reliability guarantees and their important impact on latency.

[2]We note that for companies operating in the EU market, the storage of pseudonymous information for user identifiers can help comply with the requirements of the EU's General Data Protection Regulation [HEE18].

- The first layer, the **User Anonymizer** (UA) is responsible for hiding the identity of the user by replacing it with a pseudonymous identity. It is able to see the IP address and the identifier of the user but it is not able to see the identifiers of the items sent by or returned to this user.
- The second layer, the **Item Anonymizer** (IA) is the one that directly interacts with the LRS. It is the only layer able to access items identifiers in the clear, but it is not able to access user identifiers or IP addresses. It can map actual item identifiers as used in the application's catalog to pseudonymous identifiers used by the LRS, and reversely.

Both layers can support an arbitrary number of enclave instances. All enclaves from the same layer are provisioned with the same secrets, but they do not need to share a common mutable state. This is key in enabling the proxy service to horizontally scale and handle more requests.

Protection from network inference. Attacking an enclave is not the only way the adversary may attempt to break unlinkability. As we consider it may observe communications with the LRS in the clear, the adversary could monitor the series of interactions that occur between the user and the UA layer, between the UA and the IA layers, and finally between the IA layer and the LRS. It could, eventually, link a specific IP address and both the pseudonymous user identifier and items used for the actual request.

`PProx` protects against such attacks by shuffling communication for anonymizing requests of multiple users between the UA and IA layers. We make the assumption that the system is under a flow of requests of sufficiently high volume (*e.g.*, 50 per second in our evaluation). Redirections only happen after a configurable number of requests have been buffered, and these requests are sent in a randomized order. The adversary cannot, as a result, determine precisely which final request sent to the LRS in the clear correspond to a specific incoming request to the UA. The same applies to responses sent back from the LRS to the user side. The use of buffering introduces a queuing delay to every request but this delay does not prevent from achieving overall latencies of at most a few hundred milliseconds, as required for the user to consider the system interactive [ABC14].

6.4 `PProx` protocol design

We detail in this section the `PProx` protocol, from the interception of requests at the application side to their handling by the proxy service, and their final processing by the

u	User identifier
$i_1, ..., i_n$	Item identifiers
UA	User Anonymizer, 1st layer of the proxy service
IA	Item Anonymizer, 2nd layer of the proxy service
$\mathtt{pk}_{\mathrm{UA}}$	Public key of User Anonymizer layer
$\mathtt{sk}_{\mathrm{UA}}$	Private key of User Anonymizer layer
$\mathtt{pk}_{\mathrm{IA}}$	Public key of Item Anonymizer layer
$\mathtt{sk}_{\mathrm{IA}}$	Private key of Item Anonymizer layer
$\mathtt{k}_{\mathrm{UA}}$	Permanent symmetric key of User Anonymizer layer
$\mathtt{k}_{\mathrm{IA}}$	Permanent symmetric key of Item Anonymizer layer
$\mathtt{k}_u$	Temporary symmetric key generated by user u
$\mathbf{enc}(x, \{\mathtt{p}\|\mathtt{s}\}\mathtt{k})$	asymmetric encryption of x using public/private key k
$\mathbf{det\_enc}(x, \mathtt{k})$	deterministic symmetric encryption of x using key k
S	Size of the shuffling buffer

Table 6.1: Notations

LRS. We use the notations listed in Table 6.1. We focus on the protocol in this section and discuss its implementation in Section 6.7. We analyze its security in Section 6.5.

6.4.1 Provision and use of cryptographic material

Each of the UA and IA layers is composed of a number of SGX enclaves, All enclaves in a given layer run the same code and are provisioned with the same secrets. Enclaves in the UA layer are provisioned with private key $\mathtt{sk}_{\mathrm{UA}}$ and a permanent symmetric key $\mathtt{k}_{\mathrm{UA}}$. Enclaves in the IA layer are provisioned with $\mathtt{sk}_{\mathrm{IA}}$ and $\mathtt{k}_{\mathrm{IA}}$. Provisioning the same keys to all enclaves in a layer is necessary to enable stateless load balancing of incoming requests, simplifying the implementation of horizontal scaling. New enclaves are attested upon their bootstrap before being provisioned with these keys. The two types of keys serve complementary purposes:

- Public/private key pairs enable the user-side library to encrypt information for exclusive visibility by one of the two layers. For instance, the user identifier should only be visible in the clear by the UA layer. The user-side library intercepts the cleartext request and transforms u into $\mathbf{enc}(u, \mathtt{pk}_{\mathrm{UA}})$ so that only UA enclaves may recover u from the ciphertext using $\mathtt{sk}_{\mathrm{UA}}$. However, this same ciphertext cannot be used as the pseudonym of u with the LRS, as it is the result of randomized encryption: two encryptions of the same u yield two different ciphertexts and do not allow linking to a single pseudonymous user profile.

- The permanent symmetric keys $\mathtt{k}_{\mathrm{UA}}$ and $\mathtt{k}_{\mathrm{IA}}$ are used for deterministic encryption of the users' and items' identifiers, enabling their pseudonymization. A UA enclave, accessing some user identifier u in the clear, can encrypt it such that the resulting ciphertext is the same as with another encryption of the same input. The same applies to the IA layer, where an enclave must be able to deterministically encrypt an item identifier i_x it sees in the clear. While deterministic encryption has lower security (*e.g.*, less resilience against know-plaintext attacks than probabilistic encryption), it is necessary to allow the LRS to recognize two encrypted user or item identifiers as being the same entity. We enable deterministic symmetric encryption by using the AES 256 CTR block cipher with a constant initialization vector.

In addition to these permanent keys provisioned to enclaves, `PProx` uses temporary symmetric keys generated by the user-side library in order to protect the result of a `get` request (collection of recommendations). The temporary symmetric key for a user u is denoted as $\mathtt{k}_u$. We note that, unlike for using $\mathtt{k}_{\mathrm{UA}}$ and $\mathtt{k}_{\mathrm{IA}}$ in symmetric encryption for pseudonymization, the encryption of return results uses regular randomized encryption, *i.e.*, AES with a random initialization vector.

6.4.2 Transparent REST calls redirection

The LRS offers a REST API and the user-side library intercepts unmodified calls to this API. The user-side library and the two proxy service layers modify the headers, to implement redirections, and payloads, to enable encryption. Each proxy maintains a table T storing the association between an inbound socket I (from the user-side library or from another proxy) and an outbound socket O (to another proxy or to the LRS). Responses from the LRS are forwarded backward using the same path as for the incoming request. The response is finally provided to the application by the user-side library as if it was returned by the LRS itself. We discuss the implementation and performance of redirections and the maintenance of T in Section 6.7 and focus in the following on the end-to-end lifecycle of `post` and `get` operations.

Insertion of feedback (`post` requests)

A `post` request inserts feedback about the access to an item i by a user u. There is no specific return value for this call, other than the HTTP header's success or error code

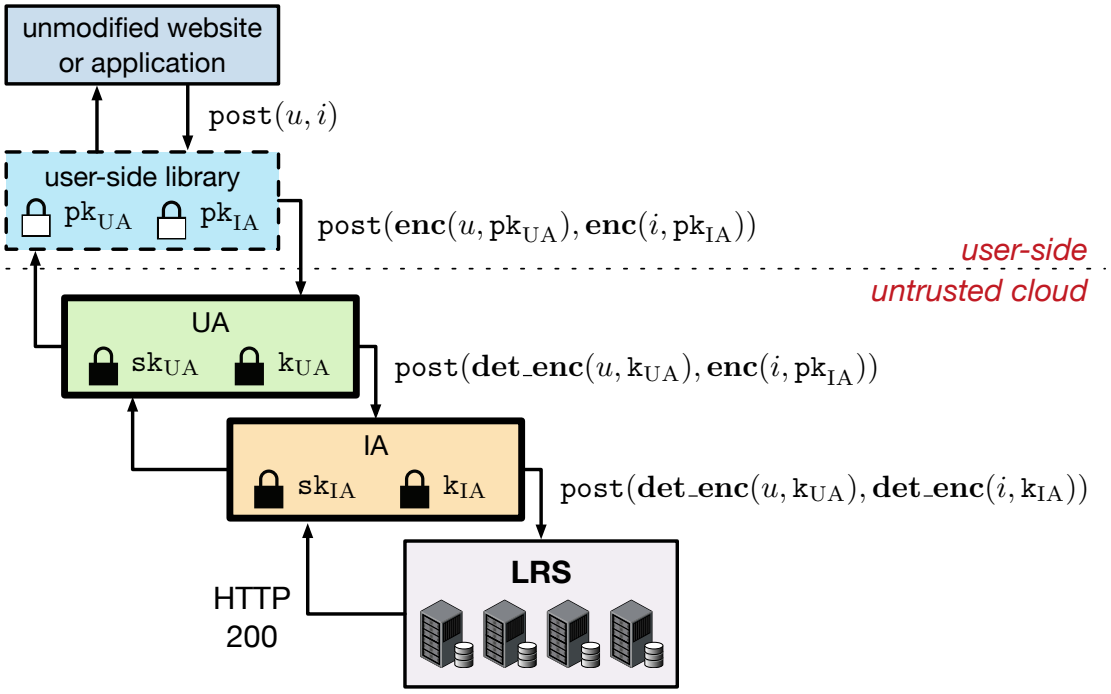


Figure 6.3: Lifecycle of a `post` request (insert feedback)

from the REST API. The end-to-end lifecycle of a `post` call is illustrated in Figure 6.3 and detailed below.

The user-side library first transforms the call $\texttt{post}(u, i)$ by encrypting the two arguments, yielding a new call

$$\texttt{post}(\textbf{enc}(u, \texttt{pk}_{\text{UA}}), \textbf{enc}(i, \texttt{pk}_{\text{IA}}))$$

that is sent to any of the enclave instances of the UA layer. This enclave decrypts u using private key $\texttt{pk}_{\text{UA}}$. It pseudonymizes plaintext u by deterministically encrypting it using $\texttt{k}_{\text{UA}}$. The resulting call

$$\texttt{post}(\textbf{det\_enc}(u, \texttt{k}_{\text{UA}}), \textbf{enc}(i, \texttt{pk}_{\text{IA}}))$$

is forwarded to one of the IA enclave instances. This enclave is able to decrypt i using private key $\texttt{sk}_{\text{IA}}$, and similarly pseudonymize the plaintext item identifier using key $\texttt{k}_{\text{IA}}$. The call containing the unlinkable information is finally forwarded to the LRS as

$$\texttt{post}(\textbf{det\_enc}(u, \texttt{k}_{\text{UA}}), \textbf{det\_enc}(i, \texttt{k}_{\text{IA}}))$$

and the response traverses back the two layers.

Collection of recommendations (get requests)

A `get` request returns a set of recommended items $(i_1, \ldots, i_n)$ tailored for a specific user u. The LRS maintains information about previous feedbacks in its database using pseudonymous item identifiers, which must be decrypted by the IA layer. This list must not be visible by the UA layer, as enclaves in the UA layer have access to the user identifier. The lifecycle of a `get` request is illustrated in Figure 6.4.

When intercepting a `get` request, the user-side library generates a temporary key $\mathtt{k}_u$ and encrypts it using $\mathtt{pk}_{\mathrm{IA}}$. This key $\mathtt{k}_u$ will be used by the IA layer to encrypt the list of recommendations and hide it from the UA, and is therefore encrypted with the IA public key $\mathtt{pk}_{\mathrm{IA}}$. The user identifier is encrypted, as for a `post` request, using the UA layer public key, yielding the call

$$\mathtt{get}(\mathbf{enc}(u, \mathtt{pk}_{\mathrm{UA}}), \mathbf{enc}(\mathtt{k}_u, \mathtt{pk}_{\mathrm{IA}})).$$

The UA enclave instance receiving this call pseudonymizes the user identifier as for a `post` request and sends the call

$$\mathtt{get}(\mathbf{det\_enc}(u, \mathtt{k}_{\mathrm{UA}}), \mathbf{enc}(\mathtt{k}_u, \mathtt{pk}_{\mathrm{IA}}))$$

to the IA layer. The IA proxy that handles this call records the identity of the source UA enclave, and sends the call $\mathtt{get}(\mathbf{det\_enc}(u, \mathtt{k}_{\mathrm{UA}}))$ to the LRS. The returned list

$$\{\mathbf{det\_enc}(i_1, \mathtt{k}_{\mathrm{IA}}), \ldots, (\mathbf{det\_enc}(i_n, \mathtt{k}_{\mathrm{IA}})\}$$

contains pseudonymized item identifiers. These identifiers are decrypted to plaintext item identifiers used by the application using $\mathtt{k}_{\mathrm{IA}}$. The recommendations list is then re-encrypted to hide it from the UA layer using the user key $\mathtt{k}_u$, yielding $\mathbf{enc}(\{i_1, \ldots, i_n\}, \mathtt{k}_u)$. The call traverses back the layers until the user-side library, which decrypts the list of recommended item identifiers using $\mathtt{k}_u$ and returns it in the clear, and transparently, to the application.

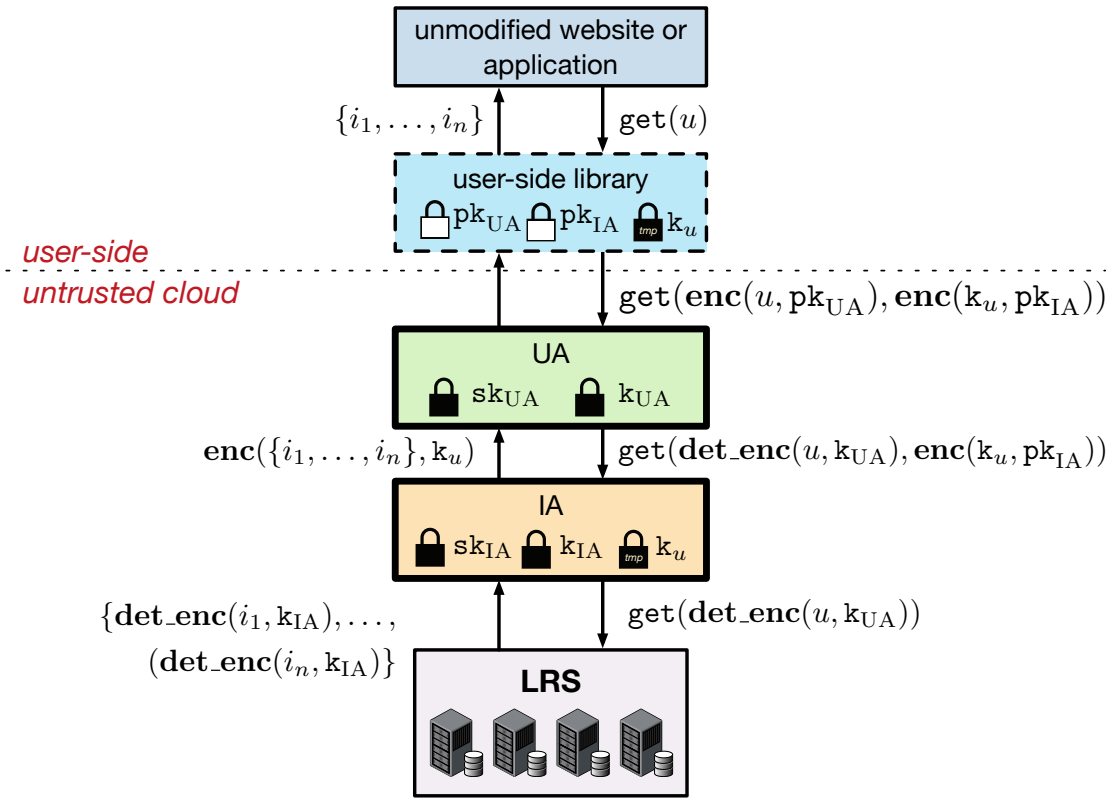


Figure 6.4: Lifecycle of a `get` req. (collect recommendations)

6.4.3 Requests and response shuffling

The pseudonymization of user and item identifiers is necessary, but not sufficient, to enable the property of User-Interest unlinkability. The adversary can observe, indeed, all network communications: between the user and UA enclaves, between UA and IA enclaves, and between IA enclaves and the LRS. By correlating in time these observations, it can relate an input request (from the user to some UA enclave) to a pseudonymized request from the IA layer to the LRS. This reveals the association between a specific user IP address and a specific set of pseudonymized item identifiers. If, in addition, the adversary was able to compromise one of the IA enclaves, it could learn the association between this user IP address and the item identifiers in the clear.

We first ensure that the adversary cannot distinguish between encrypted messages exchanged between the user-side library and the UA layer, and between the UA and IA layers. The size of all encrypted messages is constant, by using fixed-size user and item identifiers, and padding when necessary. The list of items returned by the LRS has a

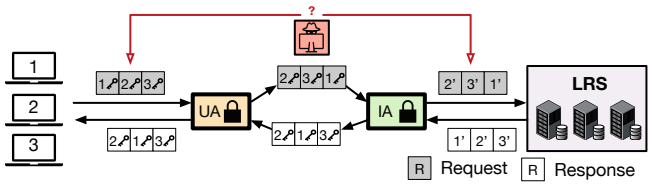


Figure 6.5: Shuffling disallows the adversary from determining which of S (here $S = 3$) incoming requests to the UA layer corresponds to a specific request sent to the LRS. The same strategy is applied to responses from the LRS.

maximal size (20 in our implementation) and we use padding to fill in missing entries. The pseudo-items used for padding are automatically discarded by the user-side library.

We implement request shuffling to protect from network inference attacks, as illustrated in Figure 6.5. Shuffling hides the direct mapping between an input request from the user to the UA layer and the redirection of this request to the IA layer. Similarly, it hides the correspondence between a response from the LRS to the IA layer, and the corresponding redirection to the UA enclave holding the user's connection. Both types of mappings are, in fact, made indistinguishable from $S - 1$ other requests, S being the size of a shuffling buffer used by the corresponding proxy (UA for requests, IA for responses). Incoming requests are buffered until S requests are received, or until a timer expires, and then sent in random order to the next stage.

Shuffling relies on the assumption that a sufficiently high amount of traffic is available for each enclave, in order to fill in the shuffling buffer before the timer expires. It is linked, therefore, with the ability to easily scale up or down each of the proxy layers, by adding and removing enclaves dynamically and ensure dynamically that sufficient redirection load applies to each of the enclaves. The size of the buffer S is a compromise between the additional latency imposed on requests and responses and the power of the attacker. This bears similarities with the principle of k-anonymity in privacy-preserving databases [LDR05, Swe02].

6.5 Security analysis

We present in this section the security analysis of `PProx`. We first present an informal proof of the User-Interest unlinkability property (§6.5.1), then analyze the impact of shuffling (§6.5.2), and finally discuss limitations (§6.5.3).

6.5.1 User-Interest Unlinkability

To break the unlinkability between a user u and an item i, the adversary must either (1) leak information from the $\texttt{post}(u, i[, p])$ message sent by u; (2) get access to items recommended by the LRS in response to a $\texttt{get}(u)$ message, in which the item i appears or (3) de-anonymize the database of the LRS. We consider the adversary defined in Section 6.2. This adversary can observe network traffic, read data stored by the LRS (*e.g.*, by running an insider attack on the machines on which the LRS runs [DCG12]) but can only break into one of the proxy service layers (*i.e.*, obtain secrets from either a UA or an IA enclave). As a reminder, the client is trusted, the adversary does not modify the data stored in the LRS and cannot break cryptographic keys. To proceed in steps, we consider the two layers of PProx separately.

Case 1: the adversary breaks a UA enclave. The adversary gains access to the following secrets: the private key $\texttt{sk}_{\text{UA}}$ used to decrypt the user identifier u contained in a transformed $\texttt{post}(u, i[, p])$ message; and the permanent key $\texttt{k}_{\text{UA}}$ used to encrypt the same user identifier u toward its storage by the LRS in pseudonymous form $\textbf{det\_enc}(u, \texttt{k}_{\text{UA}})$. We consider in the following these three (not mutually exclusive) cases: (a) the adversary intercepts the transformed $\texttt{post}(u, i[, p])$ message at a UA enclave; (b) the adversary intercepts the response to the $\texttt{get}(u)$ message containing i as a recommended item; (c) the adversary gets access to the content of the LRS database.

Case 1.(a): *the adversary intercepts a* `post` *request at a broken UA enclave.* ▷ Call $\texttt{post}(u, i[, p])$ has been transformed at the user side to $\texttt{post}(\textbf{enc}(u, \texttt{pk}_{\text{UA}}), \textbf{enc}(i, \texttt{pk}_{\text{IA}}))$. The adversary intercepts this message and knows the origin of the request. It can link the IP address to u by decrypting $\textbf{enc}(u, \texttt{pk}_{\text{UA}})$ using the stolen secret $\texttt{sk}_{\text{UA}}$. By accessing the LRS database, it may link u with $\textbf{det\_enc}(i, \texttt{k}_{\text{IA}})$ as it knows $\texttt{k}_{\text{UA}}$ and can thus decrypt $\textbf{det\_enc}(u, \texttt{k}_{\text{UA}})$. However, as long as it does not steal IA layer's secrets, the adversary cannot decrypt $\textbf{det\_enc}(i, \texttt{k}_{\text{IA}})$ and cannot, therefore, link u and i.

Case 1.(b): *the adversary intercepts the response to a* `get` *request at a broken UA enclave.* ▷ The adversary accesses a list of encrypted item identifiers containing $\textbf{enc}(\{i\}, \texttt{k}_u)$. It also knows the final destination, *i.e.*, the IP address of user u. However, it is not able to decrypt item identifiers as it does not have access to $\texttt{k}_u$, only available at the client and to the IA layer. Linking u and i would require, here again, to get secrets from IA enclaves at the same time as from UA enclaves, contradicting our fault model.

Case 1.(c): *the adversary breaks a UA enclave and also gets access to the content of the LRS database.* ▷ In this case, the adversary can de-pseudonymize user identifiers

using $\mathtt{k}_{\mathrm{UA}}$, but it is not able to de-pseudonymize items as obtaining $\mathtt{k}_{\mathrm{IA}}$ would require breaking into a second enclave in the IA layer.

Case 2: the adversary breaks an IA enclave. Breaking an IA enclave allows the adversary to gain access to the following secrets: the private key $\mathtt{sk}_{\mathrm{IA}}$ used to decrypt an item identifier i contained in a transformed $\mathtt{post}(u, i[, p])$ message; and the permanent key $\mathtt{k}_{\mathrm{IA}}$ used to pseudonymize this item identifier i as $\mathbf{det\_enc}(i, \mathtt{k}_{\mathrm{IA}})$ for use by the LRS. As for Case 1, we consider the following three (not mutually exclusive) cases: (a) the adversary intercepts the transformed $\mathtt{post}(u, i[, p])$ message at an IA enclave; (b) the adversary intercepts the response to a `get` request, containing i as a recommended item; (c) the adversary gets access to the LRS database.

Case 2.(a): *the adversary intercepts a `post` message at the broken IA enclave.* ▷ The message available to the IA layer is the result of transformations by the user-side library and by the UA layer, *i.e.*, $\mathtt{post}(\mathbf{det\_enc}(u, \mathtt{k}_{\mathrm{UA}}), \mathbf{enc}(i, \mathtt{pk}_{\mathrm{IA}}))$. The adversary can decrypt $\mathbf{enc}(i, \mathtt{pk}_{\mathrm{IA}})$ using the leaked secret $\mathtt{sk}_{\mathrm{IA}}$ to obtain i. However, it cannot know the origin of the request thanks to the shuffling of messages performed by the UA layer. By observing the LRS, the adversary can further link i with $\mathbf{det\_enc}(u, \mathtt{k}_{\mathrm{UA}})$, having access to permanent key $\mathtt{k}_{\mathrm{IA}}$. However, as long as it does not simultaneously break one of the UA enclaves, the adversary cannot decrypt $\mathbf{det\_enc}(u, \mathtt{k}_{\mathrm{UA}})$ and cannot, therefore, link u and i.

Case 2.(b): *the adversary intercepts the response to a `get` request at the broken IA enclave.* ▷ The adversary accesses a list of encrypted item identifiers containing $\mathbf{det\_enc}(i, \mathtt{k}_{\mathrm{IA}})$. It can, therefore, decrypt i using the leaked secret $\mathtt{k}_{\mathrm{IA}}$. However, thanks to message shuffling, the adversary is not able to learn for which user (IP address) the response is making it unable to link u and i.

Case 2.(c): *the adversary breaks an IA enclave and also gets access to the content of the LRS database.* ▷ The adversary does not have access to the permanent key $\mathtt{k}_{\mathrm{UA}}$, held by UA enclaves. It cannot, therefore, decrypt pseudonymous user identifiers in the LRS databases, preserving unlinkability between u and i.

In summary, even if it breaks one of the UA enclaves or IA enclaves, an adversary cannot break user-interest unlinkability despite getting privileged access to the cloud infrastructure and despite actively observing network activity.

6.5.2 Impact of Shuffling

We analyze the impact of shuffling as described in §6.4.3. UA proxy instances send requests to the IA layer in randomized batches, on the way from the user to the LRS. Each batch contains S or more requests (the latter case only decreases the power of the adversary). From the LRS to the user, IA proxy instances do the same towards the UA layer.

We first consider a single proxy instance per layer, and the user-to-LRS path. For a given time window, let us denote the set of messages output by the UA layer as out_{UA}, and the set of messages output by the IA layer as out_{IA}. Let us further assume that the adversary is interested in linking an incoming client request R to the related message R' reaching the LRS. Packets are encrypted and of the same size and, therefore, all outbound packets from the UA layer to the IA layer are equally likely to correspond to R. The odd for the attacker to correctly "guess" the correct outbound packet given an inbound packet from the client is $\frac{1}{|out_{\text{UA}}|} = \frac{1}{S}$. Note that the same applies for responses from the LRS going back towards users.

We now factor in horizontal scaling, *i.e.*, a varying number of proxy instances in each layer. On the way from the user to the LRS, the number of instances in the UA layer does not impact unlinkability, as the adversary can observe the origin (IP address) of requests to any of the instances. We denote as I the number of instances in the IA layer. The horizontal scaling of I improves unlinkability: the probability to select the correct outbound message R' for an inbound message R becomes $\frac{1}{|out_{\text{UA}}|\times I} = \frac{1}{S\times I}$. From the LRS to the user, the number of IA layer instances has no impact, and the probability for the attacker to rightly guess that a response from the LRS is for a specific IP is $\frac{1}{S\times U}$ where U is the number of UA layer instances.

6.5.3 Limitations

Assumption on traffic. The effectiveness of shuffling depends on our assumption that there is sufficient traffic. In certain cases, *e.g.*, for unpopular websites or for some given periods of times (*e.g.*, at night time), this assumption may not hold for a given application. In this case, an adversary could break the unlinkability between a user and an item if, and only if, it successfully steals secrets from the IA layer in addition to timing network requests. Such an attack is difficult to orchestrate and may be of little interest for low-traffic applications. Possible mitigation would be for the RaaS provider to leverage multi-tenancy, *i.e.*, use the same proxy layer for multiple applications, thereby increasing

the minimum traffic. This comes, however, with increased risks in case an enclave is broken, as secrets for multiple applications could be stolen at once.

Disabling item pseudonymization. In PProx, we send pseudonymous item identifiers to the LRS by default. For a large fraction of recommendation algorithms, and in particular those based on collaborative filtering, the use of pseudonymous items has no impact and is recommended for increased privacy. For algorithms that would need item identifiers in the clear, *e.g.*, for recommendations based on the semantics of the items [LDGS11], it is easy to disable the pseudonymization of items, by using i directly instead of $\mathbf{det\_enc}(i, \mathtt{k}_{\mathrm{IA}})$ for calls to the LRS. This would have, however, an impact on our provided security properties. Accepting that items be sent in clear requires, indeed, to lower down our assumed adversary to still preserve unlinkability between users and their interests. This is an example of the privacy-utility tradeoff: disabling item pseudonymization means unlinkability is preserved if and only if UA enclaves are not broken.

6.6 Integration and Reproducibility

We integrate PProx with a representative LRS, the *Universal Recommender* [Actb] (UR), initially developed for Apache Mahout and the `prediction.io` frameworks and integrated with Harness [Acta], an open-source machine learning platform. UR implements collaborative filtering based on the Correlated Cross-Occurrence (CCO) algorithm [Fer17]. CCO aggregates indicators (in our setup, feedback on the access to items) and builds profiles allowing to predict users' interests based on the history of other profiles with high similarity.

Harness uses several modules to support the UR model construction and the generation of predictions. A MongoDB database persists engine-related data and inputs pending processing (*i.e.*, feedback received via `post` requests). UR uses an `elasticsearch` instance to persist the recommendation mode, and periodic runs of Apache Spark for rebuilding this model including new inputs fetched from MongoDB. Harness frontend modules provide a REST API allowing to query the model and return JSON-encoded recommendations. These frontend modules handle the most significant part of the load. All modules can scale horizontally by adding new instances. We further detail the performance and scaling of Harness supporting UR in our evaluation (§6.8.2).

6.6.1 Workload injection and stub LRS

We built an HTTP load injector based on the high-performance `loadtest` library [Fer19] for `node.js`. The injector issues REST API calls and times their execution. When testing `PProx` in isolation from Harness, we use a stub service with the `nginx` high-performance HTTP server to serve a static payload of the same size as Harness recommendations lists.

6.6.2 Experimental reproducibility

We target the experimental reproducibility of our results through the use of an "everything-as-code" approach. All components (`PProx`, Harness, our workload injector, and `nginx`) are deployed as Docker containers in a cluster managed with MaaS [Can] and running Kubernetes [Bre15] v1.15, deployed using Kubespray v2.12.3. Since support for Intel SGX is yet to be integrated into the main version of Kubernetes we used the Kubernetes Device Plugin for Intel SGX developed by Vaucher et al. [VPF+18]. The deployment and configuration of all containers composing the system rely on charts for the Helm [Clo] package manager. We implement horizontal scaling of `PProx` proxy layers and of all Harness modules using Kubernetes integrated load balancing mechanisms (`kube-proxy` module). We collect logs in a systematic fashion using `fluentd` [Flu] and store them in a MongoDB instance separate from the one used by Harness. Experiment are described by Jupyter [Pro] notebooks in order to systematize deployment, orchestration and analysis of experimental results, and allow other researchers to reproduce them.

6.7 Implementation

We focus in this section the implementation of the privacy-preserving proxy service running in SGX enclaves. The implementation of the user-side library in Javascript and its integration into a webpage is straightforward, therefore we do not detail it in this section.

The proxy service must be able to support numerous concurrent requests. This is particularly challenging as (1) part of the proxy logics resides in SGX enclaves and (2) this logic must perform CPU-intensive cryptographic computations. In addition to a high level of concurrency, the proxy design must target fairness in the processing of requests, in order to control service time tail latency. This requires ensuring that no request gets delayed arbitrarily more than the delay that shuffling already introduces.

Scheduling the processing of requests should not introduce, on the other hand, significant synchronization overheads.

The proxy service is implemented in C++ using the Intel SGX SDK [Int20b]. Cryptographic operations use Intel's OpenSSL SGX port [Int20a], using RSA for asymmetric encryption and AES-CTR mode for symmetric encryption. We use a constant initialization vector (IV) for deterministic encryption (user and item pseudonymization). For regular encryption of data to and from the client, we use a randomly-generated IV that is stored temporarily in the enclave memory. Data from/to the client and from/to the LRS is structured in JSON, and the encrypted content is handled and stored in the `base64` format. The implementation is split in two parts, server and data processing, which we detail below.

Server The *server* runs outside of SGX enclaves and is identical for the UA and IA layers. It (i) handles connection requests and schedules their processing, implementing shuffling, and (ii) is in charge of receiving and sending packets. The server is the only component that performs system calls with the local OS: data processing enclaves only process data in memory that has been prepared by the server.

We adopt an event-driven approach to the scheduling and handling of incoming requests. The server runs as a single thread listening to incoming connection requests notification using the `epoll()` data structure and associated system calls of the Linux kernel. Incoming connections' file descriptors are pushed into a queue, to be consumed in order[3] by the pool of data processing threads. We use a lock-free, scalable concurrent queue implementation by Desrochers [Des20].

The server thread maintains table T, the routing table for pending requests, as a map from outbound file descriptors to inbound file descriptors (sockets). When the `epoll()` call raises an event for a file descriptor f, the server thread can lookup T to establish the corresponding return path.

Table T is also used for implementing request shuffling. When the number of elements in T reaches S or when the timer expires, the server enqueues all pending requests in a randomized order into the shared concurrent queue. Note that the size of T should be

[3]The order of notifications across several `epoll_wait()` system calls follows the real-time order of requests reception, except for requests received between calls that may be ordered arbitrarily. The number of such requests when the system has not reached saturation is limited and the processing of requests is, in practice, very close to the order of their reception.

larger than S in order to avoid dropping incoming requests between the reaching of the threshold and the processing of the requests. We stress that the server only processes encrypted content without the possibility of accessing it in the clear: clients' identities, keys, IVs, and data are stored inside the enclave memory.

Data processing The *data processing* part of the proxy is supported by a pool of thread running in the SGX enclave[4]. Each data processing thread dequeues work from the tail of the shared concurrent queue. For each processed packet, a thread (i) parses it (HTTP headers and JSON payloads); (ii) performs cryptographic operations as detailed in Section 6.4 and (iii) forges a new packet to forward to the other proxy layer, to the LRS, or back to the client. We implemented a lightweight JSON parser inside the enclave, able to retrieve and/or update JSON fields in place and with minimal copy overhead. An in-memory key-value store in the EPC (Enclave Page Cache) holds the information necessary for handling requests responses on their way back from the LRS.

6.8 Evaluation

We evaluate `PProx` using the reproducible experimental setup presented in the previous section. All our deployments are performed on a cluster of 27 nodes. Each node is an Intel Next Unit of Computing (NUC) Kit with a 2-core 3.50 GHz Intel i7 processor and 32 GB of RAM, recommended by Intel to experiment with SGX.

Our evaluation aims at answering the following research questions: (1) What is the impact of each of the privacy-enabling features of `PProx` (encryption, use of SGX, and request shuffling) on service latency? (2) How does the performance of Harness equipped with `PProx` compare to an unprotected deployment? (3) Is `PProx` able to scale to handle larger Harness deployments, and what are the comparative costs of the two sub-systems?[5]

[4]We use a thread pool size equal to the number of cores in our evaluation with a small number of cores, but deployments on a large multicore CPU could use one less thread in the pool than the number of cores and pin the server thread to the remaining core to reduce scheduling overheads.

[5]We choose not to evaluate `PProx` against other privacy-preserving approaches. Indeed, cryptography-based solutions have a latency above 10 seconds on a high-end server CPU, while `PProx` has a worst-case latency under one second using commodity hardware. Besides, differentially private and decentralized solutions both require complex client-side code, and degrade accuracy due to added noise. These performance differences along with the code unavailability lead us to favor a thorough evaluation of `PProx`.

We answer question (1) through a series of micro-benchmarks with `PProx` connected to a stub server. We answer questions (2) and (3) through macro-benchmarks of `PProx` connected to Harness, with increasingly large deployments.

Metrics and workload. Our primary evaluation metric is the distribution of round-trip service latencies, as measured by workload injector instance(s). When measuring the performance of a given configuration with an increasing number of requests per second (RPS), we present results up to the last value measured before reaching saturation (*i.e.*, where latencies increase drastically due to congestion). This allows measuring the supported workload under acceptable conditions rather than the peak throughput, which comes at the price of very high latencies and is, therefore, of little interest in our context. We run each experiment (*i.e.*, for each configuration and RPS pair) 6 times and report the aggregated distribution of round-trip service latencies.

The target Service-Level Objective (SLO) for round-trip service latency depends on the nature of the application or website using RaaS services. As a rule of thumb, we consider in this evaluation that a median latency below 300 ms (not accounting the latency to and from the data center hosting the RaaS services) and never exceeding twice that value should comply with typical SLOs for online services [Sha12]. For instance, Google representatives reported back in 2006 that search results displayed in more than 500 ms resulted in drops of 20% in traffic [Goo06].

We use the MovieLens dataset `ml-20m` [Gro, HK15b] as our experimental workload. This dataset is classically used for the evaluation of recommender systems. It contains feedbacks (ratings and free-text reviews) from users for movies on the collaborative MovieLens website. We use the years 2014 and 2015 as a source of feedback, corresponding to 562,888 for 17,141 different movies made by 7,288 different users. In all of our experiments, we proceed in two phases: we inject feedback for one minute and trigger the training phase of UR (using Apache Spark) in a first phase, and collect recommendations for a duration of 5 minutes in a second phase. Note that: (1) We do not report on the quality of recommendations. This is an orthogonal concern for `PProx` that depends on the LRS. Recommendations are strictly the same as when using UR in Harness directly. (2) We focus on reporting the performance of `get` requests, as these are the more time-sensitive and costlier in terms of encryption and payload.[6] (3) We trim the first and last

[6]We evaluated the costs of `post` requests and these systematically follow the same trends as for `get` requests, with only marginally lower latencies.

15 seconds of each measurement period to avoid perturbations linked with the warm-up and slow-down of injection.

	§	Fig.	Enc.	SGX	S	UA	IA	RPS
m1	6.8.1	6.6	✗	✗	✗	1	1	250
m2	6.8.1	6.6	✓	✗	✗	1	1	250
m3	6.8.1	6.6, 6.7	✓	✓	✗	1	1	250
m4	6.8.1	6.6	★	✓	✗	1	1	250
m5	6.8.1	6.7	✓	✓	5	1	1	250
m6	6.8.1, 6.8.1	6.7, 6.8	✓	✓	10	1	1	250
m7	6.8.1	6.8	✓	✓	10	2	2	500
m8	6.8.1	6.8	✓	✓	10	3	3	750
m9	6.8.1	6.8	✓	✓	10	4	4	1000

Table 6.2: Micro-benchmark configurations.
"§" and "Fig." resp. denote the section(s) and figure(s) in which the configuration is used. "Enc." stands for the use of encryption, with ★ denoting that item pseudonymization is disabled. "S" is the shuffling parameter, "UA" and "IA" the number of nodes in each proxy service layer, and "RPS" the maximal amount of Requests Per Second supported by this configuration without throttling.

6.8.1 Micro-benchmarks

Our micro-benchmarks connect the `PProx` proxy service to the `nginx` stub returning static recommendations. We consider the configurations listed in Table 6.2, with various configurations of `PProx` allowing to analyze the contribution of each security-enabling feature (use of encryption, use of SGX enclaves, use of requests shuffling) in configurations `m1-m6` and the scalability of the proxy service in configurations `m6-m9`. We use one (for `m1-7`) or two (for `m8-9`) injector nodes and increments of 50 RPS (for `m1-6`, using a single instance in each proxy layer) or 250 RPS (for `m6-m9`, when analyzing scalability). The single `nginx` server is not a bottleneck: direct requests from the injector(s) to the stub have a median latency of 1 to 2 ms and scale well over 1,000 RPS.

Dissecting the impact of privacy features

Figure 6.6 presents the distribution of latencies when adding each of the security-enabling features of `PProx` one by one, except shuffling that we evaluate separately. We emphasize that reported values are the requests round-trip time, *i.e.*, requests traverse the UA and IA layer twice, once in each direction. We can observe that the added cost of encryption is slightly higher than the cost of using SGX enclaves. The use of SGX enclaves introduces 2 to 5 ms additional median or maximal latency, about half as much as adding encryption.

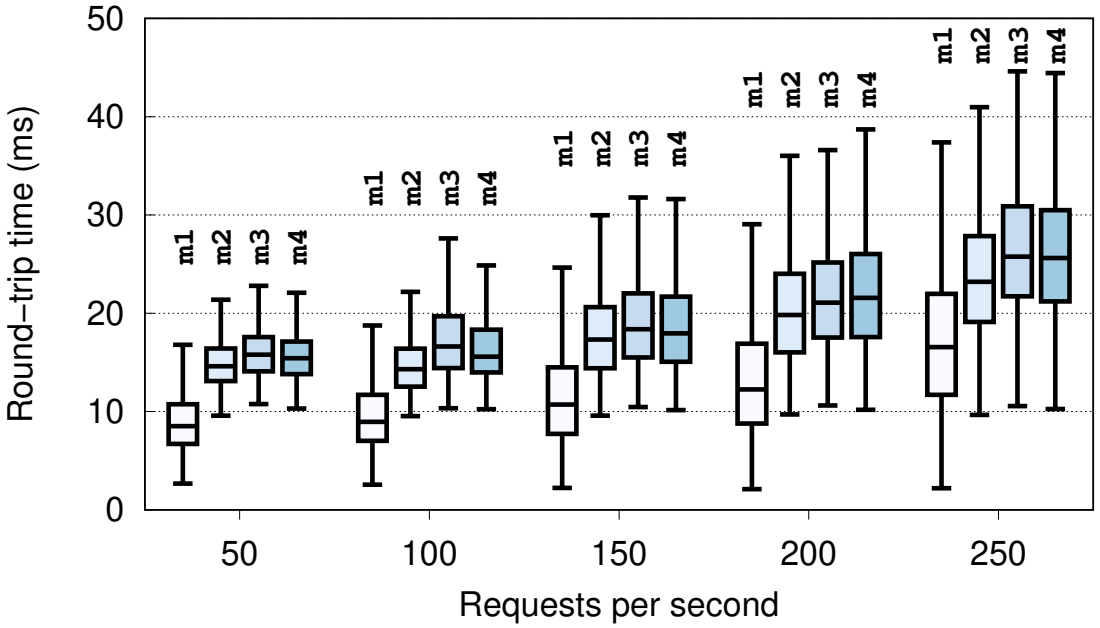


Figure 6.6: Performance of the proxy service with no security-enabling feature (`m1`), when adding encryption (`m2`), and when adding the use of SGX enclaves (`m3`); Impact of disabling item pseudonymization (`m4`).

We also disable in configuration `m4` the use of pseudonymization for item identifiers, as discussed in §6.5.3. The impact is negligible, confirming that using pseudonymous item identifiers can remain the default unless explicitly required by the recommendation algorithm.

Figure 6.7 compares the performance of a configuration with no shuffling (`m3`, same as in Figure 6.6) with configurations using shuffling. The impact of shuffling depends, unsurprisingly, on the number of requests received per second, impacting the time required to fill the buffer and send requests in a random order to the next stage (in both directions, from the UA to the IA, and from the IA to the UA). With a value of $S = 5$ and low throughput of 50 RPS, latency remains within usable boundaries for building an interactive service (at most a few hundred milliseconds) but can be too high for most SLOs when $S = 10$. With a larger number of requests per second, median round-trip service latency remains well below 200 ms in both cases. This suggests that the `PProx` proxy should not be over-provisioned (*i.e.*, using too many proxy instances for a given workload) to avoid the risk of high buffering latency. The value of S and the number

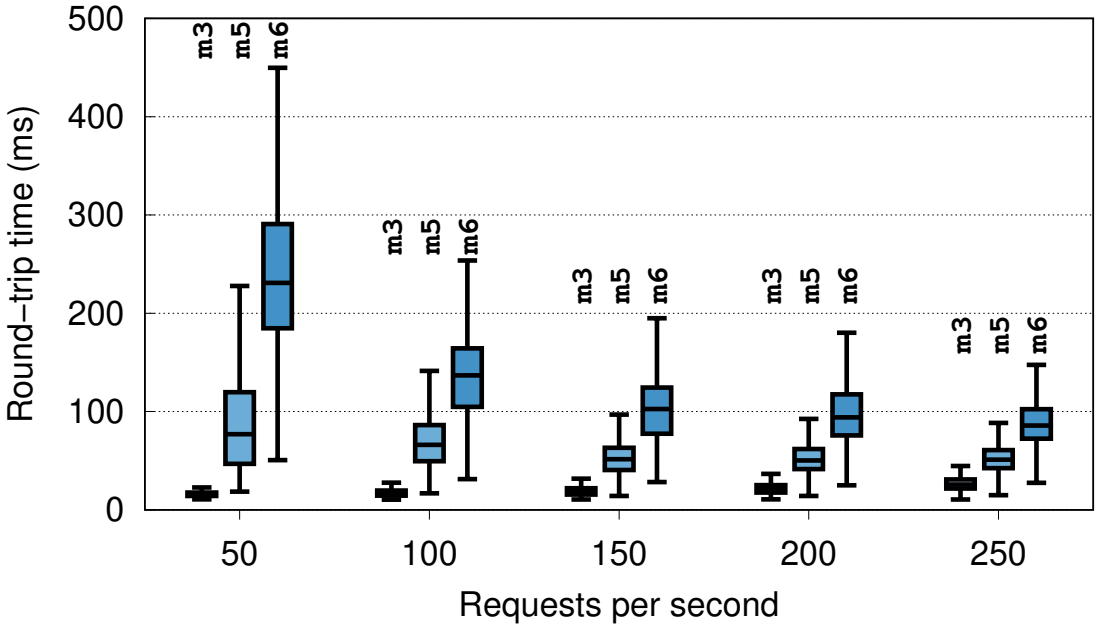


Figure 6.7: Impact of shuffling: reference configuration with no shuffling (`m3`), and with $S = 5$ (`m5`) and $S = 10$ (`m6`).

of proxy instances could be tuned elastically at run time to achieve automatically a compromise between latency and privacy, an optimization that we leave to future work.

`PProx` proxy service scaling

We finally evaluate the ability of the `PProx` proxy to scale and handle higher throughputs, starting from the complete configuration `m6` with all features and $S = 10$ from our previous experiment and using only one instance per proxy service layer. We report the results in Figure 6.8. Note that starting from this figure and for the rest of this section we switch the ordinates to a logarithmic scale for readability.

Using more proxy instances in each layer allows supporting increasing amounts of requests, *i.e.*, each additional pair of UA and IA proxy instances enables an additional 250 RPS without reaching saturation. With 4 instances of each proxy, `PProx` can offer round-trip latencies that are consistently under 200 ms for 1.000 RPS[7]. We also confirm

[7]We emphasize that the NUCs used in our evaluation only feature two cores and mobile-grade CPUs; we expect the supported throughput to also scale vertically using server-grades CPUs with support for SGX.

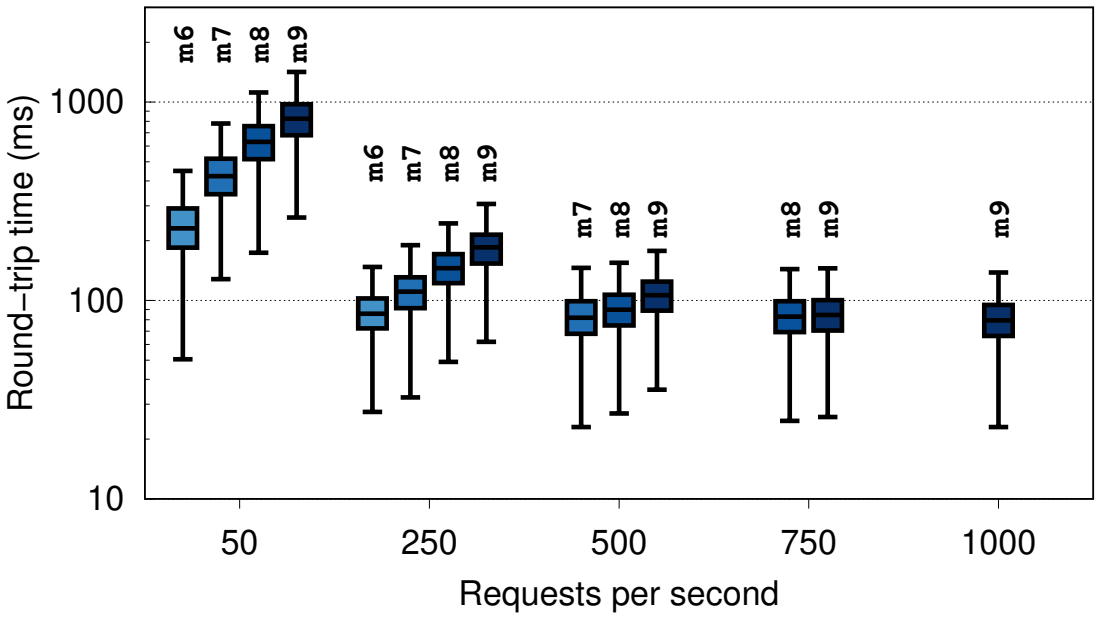


Figure 6.8: Scalability of `PProx` using 1 (`m6`) to 4 (`m9`) instances in each proxy layer (2 to 8 nodes), using all privacy-enabling features and $S = 10$.

the observation made in the previous subsection: when using an over-provisioned system (*e.g.*, `m7-9` with 50 RPS or `m9` with 250 RPS) latencies due to request shuffling may become too high to comply with the recommendation service SLO.

6.8.2 Macro-benchmarks: `PProx` with the Harness LRS

We deploy `PProx` and Harness using the configurations listed by Table 6.3. Configurations `b1-4` are for Harness deployed alone. They serve as a baseline. We vary the number of Harness front-end services from 3 to 12, and use 4 nodes for support services (three for Elasticsearch, one for MongoDB and Apache Spark). The front-end service is the main source of load for serving requests and these 4 support nodes are necessary and sufficient in all configurations. This translates to Harness configurations of 7 to 16 nodes.

Figure 6.9 presents Harness baseline performance. As previously, we present round-trip service latency for each configuration before reaching saturation. For instance, configuration `b3` with 13 nodes can serve 750 RPS with sub-second latency but saturates with 1.000 RPS. The service time latencies of Harness are representative of the type of algorithm used, that require non-trivial reads to a shared database and complex

	Fig.	Enc.	SGX	S	UA	IA	LRS	RPS
				–baseline configurations: only LRS–				
b1	6.9	✗	✗	✗	✗	✗	7: 3+4	250
b2	6.9	✗	✗	✗	✗	✗	10: 6+4	500
b3	6.9	✗	✗	✗	✗	✗	13: 9+4	750
b4	6.9	✗	✗	✗	✗	✗	16: 12+4	1000
				–full configurations: proxy service and LRS–				
f1	6.10	✓	✓	10	1	1	7: 3+4	250
f2	6.10	✓	✓	10	2	2	10: 6+4	500
f3	6.10	✓	✓	10	3	3	13: 9+4	750
f4	6.10	✓	✓	10	4	4	16: 12+4	1000

Table 6.3: Macro-benchmark experimental configurations.
"Fig." denotes the figure using the configuration. "Enc." stands for the use of encryption, S is the shuffling parameter, UA, IA and LRS are the number of nodes allocated to the proxy service layers and the LRS (front-end + support nodes). "RPS" is the maximal throughput achievable with this configuration without throttling.

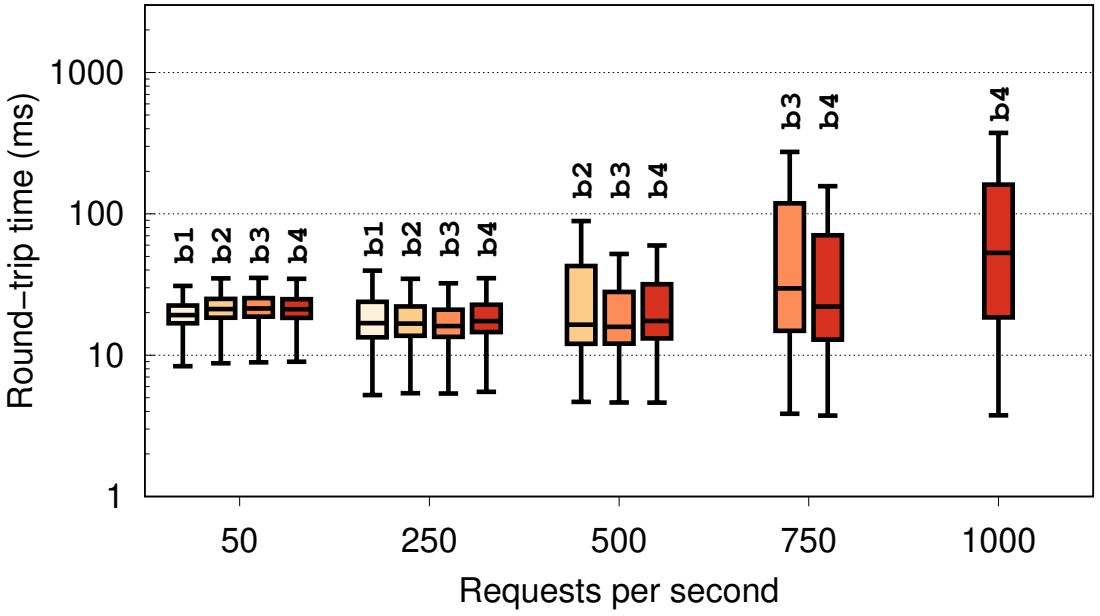


Figure 6.9: Baseline performance of the Harness LRS

(pre-built) user models to generate recommendations. These service times are below 100 ms in all configurations for low to moderate throughput (up to 500 RPS) and the spread of the distribution becomes wider for higher throughput, with peak service times around 300 ms for the largest configuration `b4` under 1.000 RPS.

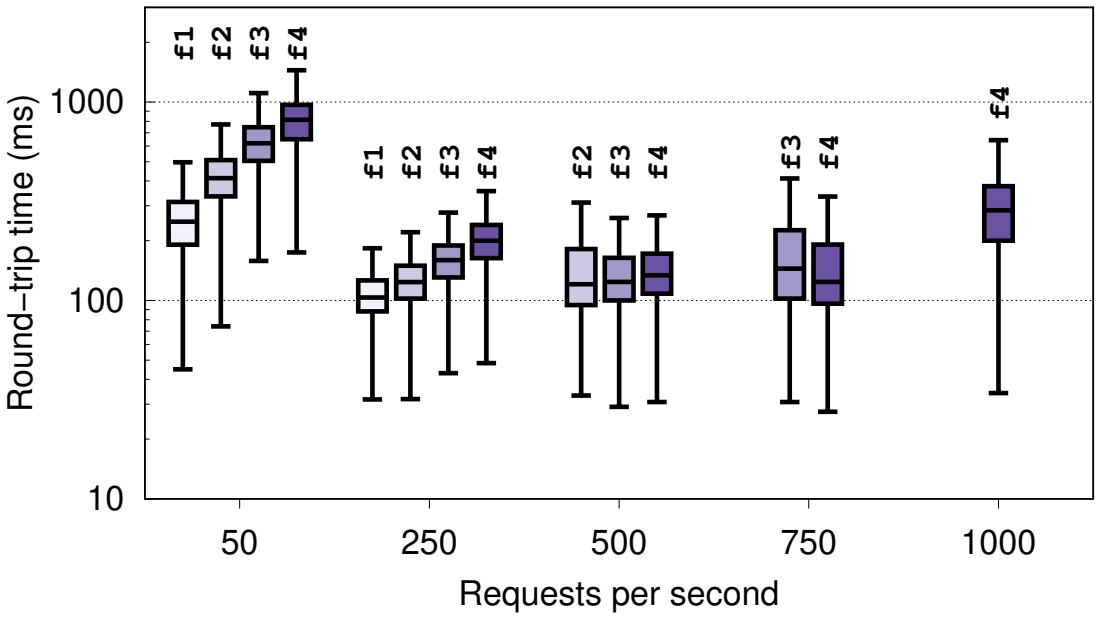


Figure 6.10: Performance of Harness when used in combination with PProx with increasingly large deployments

Figure 6.10 finally presents the performance of the complete integrated system, with PProx using 2 to 8 nodes and Harness using 7 to 16 nodes. These configurations f1-4 correspond to the combination of previously-detailed configurations m6-9 for PProx and b1-4 for Harness, supporting the same multiple of 250 RPS as maximal throughput prior to saturation. These configurations include all privacy-enabling features of PProx and use $S = 10$. The infrastructure cost of PProx ranges, therefore, from 30% (f1) to 50% (f4) additional nodes compared to privacy-unprotected Harness. Latencies are, as expected, the sum of latencies observed in Figures 6.8 and 6.9. With 50 RPS the impact of request shuffling is important, in particular for configurations f2-4. This is intrinsic to the need to prevent network observation attacks, as previously observed, and a lower value of S would shift the privacy-performance tradeoff towards the latter. For workloads of 250 to 750 RPS, however, overall latencies are systematically below 300 ms, with a median between 100 and 200 ms. With 1,000 RPS the max service time increases to 450 ms but median latency remains below 200 ms. We can contrast these latency results with the experimental evaluation of privacy-preserving recommendation algorithms based on encrypted processing [BVK+12, BVKD13, WTAR19] yielding latencies of several seconds.

6.9 Conclusion

Recommender systems usually complement streaming solutions to retain users on their website or application. Yet, they pose a serious threat to privacy, as user profiles are established based on the watching history. A few privacy-preserving recommender systems exist. However, all of them use specific recommendation algorithms, require to install a heavy code layer at the client side, and either provide a low performance with several seconds delay, or low accuracy because of added noise. To tackle these issues, we presented PProx, a system for efficient, reliable and scalable privacy preservation fitting the requirements of RaaS. PProx contributes a privacy-preserving proxy service that prevents the disclosure of the link between individuals and their interests. The security guarantees of PProx hold even in the presence of a powerful attacker able to use recently-documented side-channel attacks on SGX enclaves, and observing all network traffic in the cloud. In contrast with previous work, privacy-preservation with PProx is not specific to a recommendation algorithm and does not require complex deployment of code or state at the users' side.

We integrated PProx with the Harness universal recommendation engine and evaluated it on a 27-node cluster. Our results indicate its ability to withstand a high number of requests with low end-to-end latency, scaling up to match the workload of recommendations. The typical latency overhead is below 100ms (compared to several seconds for similar systems), and PProx only requires 30% to 50% additional nodes to provide the recommendation service.

One can explore the use of PrivaTube and PProx foundations (*i.e.*, the use of HTTP proxies in TEEs) for privacy preservation in generic online services accessed through REST APIs.

Summary

`PProx` does not impact recommendations accuracy, supports arbitrary recommendation algorithms, and has minimal deployment requirements. Its design leverages an elastically scalable network of proxies to transparently pseudonymize users over a fleet of Intel SGX-enabled machines. `PProx` privacy guarantees are robust to even the corruption of one of the SGX enclaves.

Chapter 7

Conclusion and further directions

I have not failed. I've just found 10,000 ways that won't work.

— Thomas A. Edison

This chapter concludes the thesis, summarizes the contributions and provides insight on possible future research directions.

7.1 Contributions summary

Delivering video content with a high and fairly shared Quality of Experience is a challenging task in view of the drastic video traffic increase forecasts, as live video traffic will grow 15-fold by 2022. Currently, Content Delivery Networks provide numerous servers hosting replicas of the video content and consuming clients are re-directed to the closest one. Then, the video content is streamed using HTTP Adaptive Streaming solutions. However, servers and network links often become overloaded during major events, and users may experience a poor or unfairly distributed QoE, unless more servers are provisioned. `Muslin` [DSBQL$^+$19, DSBQL$^+$18a, DSBQL$^+$18b] is a solution supporting a **high, fairly shared end-users QoE** for live streaming, while **minimizing the required content delivery platform scale**. `Muslin` leverages on MS-Stream, a content delivery solution which aggregates video content from multiple servers to offer a high QoE for its users. `Muslin` dynamically provisions servers, replicates content into them, and advertises them to clients based on real-time delivery conditions. We have used `Muslin` to replay a one-day video-games event, with hundreds of clients and several testbeds. Our results show that our approach outperforms traditional content delivery

schemes by increasing the fairness and QoE at the user side with a smaller infrastructure scale.

Efficient and scalable video streaming requires a large video delivery platform such as an edge-assisted collaborative CDN. However, access histories can reveal critical personal information, and centralized video streaming 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 streaming services. **PrivaTube** [DSBMC+19] is a **scalable** and **cost-effective** solution. 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 used 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.

Recommendation-as-a-Service enables developers to integrate personalized navigation to their websites or applications without having to master the complexity of tuning and deploying a dedicated recommendation system. All major video streaming services include a recommendation functionality. However, the generation of recommendations relies on the collection of navigation history and feedback from users. This leads to legitimate concerns regarding the privacy impact of outsourcing such sensitive data. `PProx` complements video streaming systems by providing **pseudonymous and private recommendations**. `PProx` does not impact recommendations accuracy, supports arbitrary recommendation algorithms, and has **minimal deployment requirements and performance impact**. Its design leverages a network of proxies, running on the same untrusted cloud as a legacy recommendation engine, to transparently pseudonymize users, items, as well as to hide links between the two. These proxies elastically scale over a fleet of Intel SGX-enabled machines. `PProx` privacy guarantees are **robust** to even the corruption of one of the SGX enclaves. We integrated `PProx` with the Harness recommendation engine and evaluated it on a 27-node cluster. Our results indicate its ability to withstand a high number of requests with low end-to-end latency, scaling up to match the workload of recommendations.

7.2 Further research directions

We introduce several research directions for each contribution.

`Muslin`

As `Muslin` evolved into PrivaTube, the most obvious improvements and research prospects were included in the latter. Still, a few remaining research directions were identified.

Cost model One could consider a short-term cost model improvement to `Muslin`. Its cost model could take into account scaling and network costs to further improve benefits towards infrastructure cost and cloud computing capabilities. For instance, public cloud network, storage and computing costs could be retrieved by the provisioning module to dynamically select different cloud providers depending on the region and availability.

Peer selection in P2P and edge-assisted systems A long-term generalization of `Muslin` algorithms and principles could be further leveraged for peer selection in P2P and edge-assisted streaming systems. Indeed, BitTorrent clients (and similar solutions) choose peers either at random or on past upload capabilities, which do not reflect current network and delivery conditions. These systems would thus greatly benefit from smarter source selection mechanisms.

PrivaTube

As PrivaTube is the first video streaming system to provide strong privacy guarantees with high QoE, many aspects are unexplored and could benefit from further research.

Fake requests generation Possible mid-term future work revolves around the investigation of more sophisticated fake requests generation. For instance, content access patterns over time could be taken into consideration (*e.g.*, sequential series episodes). Other ways include user profiles computation for more realistic fake requests, or even federated learning.

Proxy-less CDN implementation Another mid-term research direction is the generation of fake requests for high QoE and best-effort privacy from the client in a legacy CDN

environment. These requests could adapt according to bandwidth and buffer, similarly to DASH adaptations techniques. Fake requests rate δ could be a parameter (*e.g.*, best effort, 50%, 25%, etc.). It could rely on a trusted manifest server to register contents popularity and reply to clients requests with *samePop* contents. These improvements would only rely on a small JavaScript code overlay on top of current off-the-shelf video players.

QoE and reliability in 5G networks 5G networks are being deployed globally and raise several issues regarding video streaming. Future indoor 5G and wireless networks architectures will include heterogeneous technologies. There is currently no reliable solution enabling high-QoE video streaming over multiple heterogeneous paths. This issue has been addressed by a fellow PhD candidate. *MSS/RRLH* [LN20], a system to reliably deliver high quality and low delay videos, was proposed to tackle this challenge. It relies on edge computing resources and efficient multiple-path quality selection algorithms.

Incentive and rewarding through blockchain One of the main collaborative systems issues is users' contribution. In PrivaTube, the HTTP proxy answers to content requests within the enclave and thus enforces upload from the client. However, users could still voluntarily delete their local filesystem contents or disable their internet connection to prevent network traffic. Indeed, without incentive to contribute, many users will selfishly consume resources (*e.g.*, content or bandwidth) without contributing back to the system. To solve this issue, a few free-riding control mechanisms were designed. Some systems reward upload capacity or storage in P2P-VOD systems [WLM11, WML13]. But most mechanisms implement either direct [SRR09] or indirect [LGC+09] reciprocity schemes. For instance, SVC-TChain [RJWS+17] incentivizes good behavior in layered P2P video streaming through a triangular reciprocity scheme, called TChain [SJWH+17]. However, in PrivaTube, clients download video segments from other peers to obtain QoE improvements in addition to streaming from the public servers, and do not exclusively obtain data from peers. A reciprocity-based incentive mechanism cannot be applied to PrivaTube. There were attempts to reward users who share their resources using blockchain or cryptocurrencies. A distributed P2P system was theorized by Y. He *et al* [HLC+18], in which contributing users are automatically rewarded with any legacy cryptocurrency. Besides, social networks such as Sphere [Sph17], Steemit [Ste18] and Socialx [Soc18] use distributed ledgers and blockchains to reward contributing users with cryptocurrencies, built on top of Ethereum. Some similar implementations were made by Filecoin [Fil18] and Storj [Sto18], where users can trade disk storage against

tokens as a distributed alternative to cloud storage. To do so, they both built custom blockchain mechanisms and designed cryptocurrencies to ensure proof of retrievability and availability properties. The use of a blockchain to track users contributions (*i.e.*, upload) to the system would benefit PrivaTube, as it enables users to maintain their upload/download ratio over several devices and adds accountability.

`PProx`

Trusted Execution Environments are still being developed and integrated into commodity hardware. Their ubiquity will enable many new services to protect users' privacy and security without compromising on performance.

Automation of data pseudonymization A long-term improvement would be to use `PProx` HTTP proxies to automate data pseudonymization in systems handling sensitive user data in untrusted clouds. Several generic layers could handle the encryption and pseudonymization of data in a way that compromising one would still preserve overall unlinkability. It would require analyzing the data sent and received, and provisioning as many layers as data fields, each layer being in charge of a field.

Generic REST APIs Another long-term research project could be to leverage `PProx` foundations (*i.e.*, the use of HTTP proxies in TEEs) for privacy preservation in generic online services accessed through REST APIs. The use of efficient shuffling as in `PProx` provides differential privacy guarantees along with a low latency, unlike Tor and similar systems.

7.3 Closing remarks

These further research directions led to recruit an intern and a PhD candidate. They will both aim at expanding PrivaTube capabilities and `PProx` principles.

Video streaming is evolving quickly and faces many challenges and opportunities. We believe that our work proves strong privacy and security guarantees no longer come with low performance and QoE. Many efficient and pragmatic solutions could therefore be developed in the industry from these foundations.

We hope generalizing `Muslin`, PrivaTube and `PProx` will enable a new generation of **High-QoE Privacy-Preserving Video Streaming** services with great performance.

Bibliography

[ABC14] Ioannis Arapakis, Xiao Bai, and B Barla Cambazoglu. Impact of response latency on user behavior in web search. In *37th international ACM SIGIR conference on Research & development in information retrieval*, 2014.

[Acta] ActionML. Harness: microservice based machine learning server.

[Actb] ActionML. The Universal Recommender.

[Ado18] Adobe. Adobe HTTP dynamic streaming (HDS). `https://www.adobe.com/products/hds-dynamic-streaming.html`, 2018.

[Ado20] Adobe. Real-time messaging protocol (RTMP) specification. Technical report, Adobe, 2020.

[AGH+12] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-CDN movie delivery. In *2012 Proceedings IEEE INFOCOM*, pages 1620–1628. IEEE, 2012.

[AJCZ12] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Vivisecting youtube: An active measurement study. In *2012 Proceedings IEEE INFOCOM*, pages 2521–2525. IEEE, 2012.

[AK06] Naveen Farag Awad and Mayuram S Krishnan. The personalization privacy paradox: an empirical evaluation of information transparency and the willingness to be profiled online for personalization. *MIS quarterly*, pages 13–28, 2006.

[App18] Apple. HTTP live streaming (HLS). `https://developer.apple.com/streaming/`, 2018.

[ARM] ARM. Trustzone: system-wide hardware isolation for trusted software. `https://developer.arm.com/ip-products/security-ip/trustzone`.

[ATG+16] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L Stillwell, et al. SCONE: Secure linux containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2016.

[BAGV18] Alessio Botta, Aniello Avallone, Mauro Garofalo, and Giorgio Ventre. A user-oriented performance comparison of video hosting services. *Computer Communications*, 116:118–131, 2018.

[BBC12] Jordi Mongay Batalla, Andrzej Bęben, and Yiping Chen. Optimization of the decision process in network and server-aware algorithms. In *2012 15th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6. IEEE, 2012.

[BCE+07] Mira Belenkiy, Melissa Chase, C Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making P2P accountable without losing privacy. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 31–40, 2007.

[BCT+18] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix CDN. *ACM SIGCOMM Computer Communication Review*, 48(1):28–34, 2018.

[BEK16] Yahya Benkaouz, Mohammed Erradi, and Anne-Marie Kermarrec. Nearest neighbors graph construction: Peer sampling to the rescue. In *International Conference on Networked Systems*, NETYS. Springer, 2016.

[BFG+16] Antoine Boutet, Davide Frey, Rachid Guerraoui, Arnaud Jégou, and Anne-Marie Kermarrec. Privacy-preserving distributed collaborative filtering. *Computing*, 98(8):827–846, 2016.

[BGO19] Joeran Beel, Alan Griffin, and Conor O'Shea. Darwin & Goliath: A white-label recommender-system as-a-service with automated algorithm-selection. In *Demonstration at the 13th ACM Conference on Recommender Systems*, RecSys, 2019.

[BMD+17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[BOHG13] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46, 2013.

[BQLN17a] Joachim Bruneau-Queyreix, Mathias Lacaud, and Daniel Négru. A multiple-source adaptive streaming solution enhancing consumer's perceived quality. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 580–581. IEEE, 2017.

[BQLN+17b] Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, Jordi Mongay Batalla, and Eugen Borcoci. Ms-stream: A multiple-source adaptive streaming solution enhancing consumer's perceived quality. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 427–434. IEEE, 2017.

[BQLN+17c] Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, Jordi Mongay Batalla, and Eugen Borcoci. QoE enhancement through cost-effective adaptation decision process for multiple-server streaming over HTTP. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2017.

[BQLN+18] Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, Jordi Mongay Batalla, and Eugen Borcoci. Adding a new dimension to HTTP adaptive streaming through multiple-source capabilities. *IEEE MultiMedia*, 25(3):65–78, 2018.

[Bre15] Eric A Brewer. Kubernetes and the path to cloud native. In *Sixth ACM Symposium on Cloud Computing*, SOCC, 2015.

[BSG17] Vincent Bindschaedler, Reza Shokri, and Carl A Gunter. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment*, 10(5):481–492, 2017.

[Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4), 2002.

[BVK+12] Anirban Basu, Jaideep Vaidya, Hiroaki Kikuchi, Theo Dimitrakos, and Srijith K Nair. Privacy preserving collaborative filtering for SaaS enabling PaaS clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1):8, 2012.

[BVKD11] Anirban Basu, Jaideep Vaidya, Hiroaki Kikuchi, and Theo Dimitrakos. Privacy-preserving collaborative filtering for the cloud. In *23rd International Conference on Cloud Computing Technology and Science*, CloudCom. IEEE, 2011.

[BVKD13] Anirban Basu, Jaideep Vaidya, Hiroaki Kikuchi, and Theo Dimitrakos. Privacy-preserving collaborative filtering on the cloud and practical implementation experiences. In *Sixth IEEE International Conference on Cloud Computing*, 2013.

[BWG+16] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. Securekeeper: confidential zookeeper using Intel SGX. In *Proceedings of the 17th International Middleware Conference*, pages 1–13, 2016.

[CA07] Richard Cissée and Sahin Albayrak. An agent-based approach for privacy-preserving recommender systems. In *6th international joint conference on Autonomous agents and multiagent systems*, AAMAS. ACM, 2007.

[Can] Canonical. MaaS: Very fast server provisioning for your data centre.

[CAR17] Shujie Cui, Muhammad Rizwan Asghar, and Giovanni Russello. Privacy-preserving content delivery networks. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 607–610. IEEE, 2017.

[CC02] John Canny and John Canny. Collaborative filtering with privacy via factor analysis. In *25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2002.

[CCX+19] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. In *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019.

[CD16a] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[CD16b] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. *ACM SIGOPS Operating Systems Review*, 37(5):298–313, 2003.

[CDM+09] David R Choffnes, Jordi Duch, Dean Malmgren, Roger Guierma, Fabian E Bustamante, and Luis Amaral. Swarmscreen: Privacy through plausible deniability in P2P systems. *Technical report, Northwestern EECS*, 2009.

[CDN18] CDNPlanet. An overview of content delivery networks with pops in united states. `https://www.cdnplanet.com/geo/united-states-cdn`, 2018.

[Cis18] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. Technical report, Cisco, 2018.

[CKN+11] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, S&P, 2011.

[Clo] Cloud Native Computing Foundation. Helm: The package manager for Kubernetes.

[CPK95] Ann L Chervenak, David A Patterson, and Randy H Katz. Storage systems for movies-on-demand video servers. In *Proceedings of IEEE 14th Symposium on Mass Storage Systems*, pages 246–256. IEEE, 1995.

[CZRZ17] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18, 2017.

[Dai20] Dailymotion. The home for videos that matter. `https://www.dailymotion.com`, 2020.

[DAS] DASH Industry Forum. Guidelines for implementation: DASH-AVC/264 Test cases and Vectors. `https://dashif.org/docs/{DASH}-AVC-264-Test-Vectors-v1.0.pdf`.

[DBYEV19] Jérémie Decouchant, Antoine Boutet, Jiangshan Yu, and Paulo Esteves-Verissimo. P3ls: Plausible deniability for practical privacy-preserving live streaming. In *SRDS 2019-38th International Symposium on Reliable Distributed Systems*, 2019.

[DCG12] Adrian J Duncan, Sadie Creese, and Michael Goldsmith. Insider attacks in cloud computing. In *11th international conference on trust, security and privacy in computing and communications*, TrustCom. IEEE, 2012.

[Des20] Cameron Desrochers. Lock-free queue for C++11. `https://github.com/cameron314/concurrentqueue`, January 2020.

[DHT04] Tai T Do, Kien A Hua, and Mounir A Tantaoui. P2vod: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*, volume 3, pages 1467–1472. IEEE, 2004.

[DI18] DASH-IF. Dash-if position paper: Server and network assisted DASH (sand). `https://dashif.org/docs/SAND-Whitepaper-Dec13-final.pdf`, 2018.

[DLHC05] Chris Dana, Danjue Li, David Harrison, and Chen-Nee Chuah. Bass: Bittorrent assisted streaming system for video-on-demand. In *2005 IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4. IEEE, 2005.

[DMPQ16] Jérémie Decouchant, Sonia Ben Mokhtar, Albin Petit, and Vivien Quéma. Pag: Private and accountable gossip. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 35–44. IEEE, 2016.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[DSBMC+19] Simon Da Silva, Sonia Ben Mokhtar, Stefan Contiu, Daniel Négru, Laurent Réveillère, and Etienne Rivière. Privatube: Privacy-preserving edge-assisted video streaming. In *Proceedings of the 20th International Middleware Conference*, pages 189–201, 2019.

[DSBQL+18a] Simon Da Silva, Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, and Laurent Réveillère. MUSLIN: Achieving high, fairly shared

QoE through multi-source live streaming. In *Proceedings of the 23rd Packet Video Workshop*, pages 54–59, 2018.

[DSBQL+18b] Simon Da Silva, Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, and Laurent Réveillère. MUSLIN demo: high QoE fair multi-source live streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 529–532, 2018.

[DSBQL+19] Simon Da Silva, Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, and Laurent Réveillère. MUSLIN: A QoE-aware CDN resources provisioning and advertising system for cost-efficient multisource live streaming. *International Journal of Network Management*, page e2081, 2019.

[dSDR16] Pedro Moreira da Silva, Jaime Dias, and Manuel Ricardo. Mistrustful P2P: Privacy-preserving file sharing over untrustworthy peer-to-peer networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 395–403. IEEE, 2016.

[DTCU17] Jie Deng, Gareth Tyson, Felix Cuadrado, and Steve Uhlig. Internet scale user-generated live video streaming: The twitch case. In *International Conference on Passive and Active Network Measurement*, pages 60–71. Springer, 2017.

[DTS+19] Erika Duriakova, Elias Z Tragos, Barry Smyth, Neil Hurley, Francisco J Peña, Panagiotis Symeonidis, James Geraci, and Aonghus Lawlor. PDMFRec: a decentralised matrix factorisation with tunable user-centric privacy. In *13th ACM Conference on Recommender Systems*, RecSys, 2019.

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, TAMC. Springer, 2008.

[Fer17] Pat Ferrel. Multi-domain predictive AI or how to make one thing predict another, June 2017.

[Fer19] Alex Fernández. alexfernandez/loadtest, December 2019. original-date: 2013-06-21T23:50:01Z.

[FH07] Daniel M Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. In *8th ACM conference on Electronic commerce*. ACM, 2007.

[Fil18] Filecoin. Filecoin whitepaper. `https://filecoin.io/filecoin.pdf`, 2018.

[FKV+15] Arik Friedman, Bart P Knijnenburg, Kris Vanhecke, Luc Martens, and Shlomo Berkovsky. Privacy aspects of recommender systems. In *Recommender Systems Handbook*. Springer, 2015.

[Flu] Fluentd project. Fluentd: an open source data collector for unified logging layer.

[FMM+15] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, 2015.

[GCM+16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107, 2016.

[GDBJ10] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *4th ACM conference on Recommender systems*, RecSys, 2010.

[GEB+13] Panagiotis Georgopoulos, Yehia Elkhatib, Matthew Broadbent, Mu Mu, and Nicholas Race. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pages 15–20, 2013.

[GESM17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on Intel SGX. In *10th European Workshop on Systems Security*, EuroSec. ACM, 2017.

[GFD+14] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. Offline and online evaluation of news recommender systems at swissinfo.ch. In *8th ACM Conference on Recommender systems*, 2014.

[GHK+10] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. Lifting: lightweight freerider-tracking in gossip. In *ACM/IFIP/ USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 313–333. Springer, 2010.

[GJP12] Saikat Guha, Mudit Jain, and Venkata N Padmanabhan. Koi: A location-privacy platform for smartphone apps. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 183–196, 2012.

[GKP+17] Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, Mahammad Valiyev, and Jingjing Wang. I know nothing about you but here is what you might like. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 439–450. IEEE, 2017.

[GLS+17] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th USENIX Security Symposium*, 2017.

[Goo06] Google VP Marrisa Mayer. Presentation at Third Annual Web 2.0 Summit, 2006.

[Goo18] Google. Google cloud CDN points of presence. `https://cloud.google.com/cdn/docs/locations`, 2018.

[Goo20] Google. QUIC, a multiplexed stream transport over UDP. Technical report, Google, 2020.

[Gro] GroupLens research at the University of Minnesota. Description of the movielens `ml-20m` dataset.

[HEE18] Mike Hintze and Khaled El Emam. Comparing the benefits of pseudonymisation and anonymisation under the GDPR. *Journal of Data Protection & Privacy*, 2(2), 2018.

[Hiv] Hive Streaming. Enterprise content delivery, video distribution software. https://www.hivestreaming.com/.

[HK15a] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[HK15b] F Maxwell Harper and Joseph A Konstan. The MovieLens datasets: History and context. *ACM transactions on interactive intelligent systems (TIIS)*, 5(4), 2015.

[HLC+18] Yunhua He, Hong Li, Xiuzhen Cheng, Yan Liu, Chao Yang, and Limin Sun. A blockchain based truthful incentive mechanism for distributed P2P applications. *IEEE Access*, 6:27324–27335, 2018.

[HSH+11] Tobias Hoßfeld, Michael Seufert, Matthias Hirth, Thomas Zinner, Phuoc Tran-Gia, and Raimund Schatz. Quantification of youtube QoE via crowdsourcing. In *2011 IEEE International Symposium on Multimedia*, pages 494–499. IEEE, 2011.

[HSKHV16] Tobias Hoßfeld, Lea Skorin-Kapov, Poul E Heegaard, and Martin Varela. Definition of QoE fairness in shared systems. *IEEE Communications Letters*, 21(1):184–187, 2016.

[HWC+15] Han Hu, Yonggang Wen, Tat-Seng Chua, Jian Huang, Wenwu Zhu, and Xuelong Li. Joint content replication and request routing for social video distribution over cloud CDN: A community clustering method. *IEEE transactions on circuits and systems for video technology*, 26(7):1320–1333, 2015.

[Int20a] Intel. Intel Software Guard Extensions SSL. https://github.com/intel/intel-sgx-ssl, March 2020.

[Int20b] Intel. SDK for Intel Software Guard Extensions. https://software.intel.com/en-us/sgx/sdk, April 2020.

[IT17] ITU-T. P.1203 : Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport. https://www.itu.int/rec/T-REC-P.1203, 2017.

[KCG17] Vaibhav Kulkarni, Bertil Chapuis, and Benoît Garbinato. Privacy-preserving location-based services by using Intel SGX. In *1st International Workshop on Human-centered Sensing, Networking, and Systems*. ACM, 2017.

[KHH+18] Seongmin Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. SGX-Tor: A secure and practical tor anonymity network with SGX enclaves. *IEEE/ACM Transactions on Networking*, 26(5), 2018.

[KMSG13] Miltiadis Kandias, Lilian Mitrou, Vasilis Stavrou, and Dimitris Gritzalis. Youtube user and usage profiling: Stories of political horror and security success. In *International Conference on E-Business and Telecommunications*, pages 270–289. Springer, 2013.

[KPW+19] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. Shieldstore: Shielded in-memory key-value storage with SGX. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–15, 2019.

[KRR02] Jussi Kangasharju, James Roberts, and Keith W Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4):376–383, 2002.

[LBSR14] Kyongchun Lim, Yonghwan Bang, Jihoon Sung, and June-Koo Kevin Rhee. Joint optimization of cache server deployment and request routing with cooperative content replication. In *2014 IEEE International Conference on Communications (ICC)*, pages 1790–1795. IEEE, 2014.

[LCW+06] Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 191–204, 2006.

[LDGS11] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[LDR05] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *ACM SIGMOD international conference on Management of data*, 2005.

[LGC+09] Raul Landa, David Griffin, Richard G Clegg, Eleni Mykoniati, and Miguel Rio. A sybilproof indirect reciprocity mechanism for peer-to-peer networks. In *IEEE INFOCOM 2009*, pages 343–351. IEEE, 2009.

[LGL08] Yong Liu, Yang Guo, and Chao Liang. A survey on peer-to-peer video streaming systems. *Peer-to-peer Networking and Applications*, 1(1):18–28, 2008.

[LN20] Mathias Lacaud and Daniel Négru. Multiple-Source Streaming over Remote Radio Light Head: a pragmatic, efficient and reliable video streaming system for 5G intra-building use cases. In *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2020.

[LOH16] Wenjie Li, Sharief MA Oteafy, and Hossam S Hassanein. Streamcache: Popularity-based caching for adaptive streaming over information-centric networks. In *2016 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2016.

[LPM+17] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, et al. Glamdring: Automatic application partitioning for intel SGX. In *USENIX Annual Technical Conference*, ATC, 2017.

[LSG+17] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 557–574, 2017.

[LXZ+14] Dixin Luo, Hongteng Xu, Hongyuan Zha, Jun Du, Rong Xie, Xiaokang Yang, and Wenjun Zhang. You are what you watch and when you watch: Inferring household structures from iptv viewing data. *IEEE Transactions on Broadcasting*, 60(1):61–72, 2014.

[MBDC18] Ricky KP Mok, Vaibhav Bajpai, Amogh Dhamdhere, and KC Claffy. Revealing the load-balancing behavior of youtube traffic on interdomain links. In *International Conference on Passive and Active Network Measurement*, pages 228–240. Springer, 2018.

[MBF+17a] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. X-search: revisiting private web search using Intel SGX. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 198–208, 2017.

[MBF+17b] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. X-search: revisiting private web search using Intel SGX. In *18th ACM/IFIP/USENIX Middleware Conference*, 2017.

[MDMÖG18] Itishree Mohallick, Katrien De Moor, Özlem Özgöbek, and Jon Atle Gulla. Towards new privacy regulations in europe: Users' privacy perception in recommender systems. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, SpaCCS. Springer, 2018.

[Med19] Mediego. The turnkey solution to personalize your content in real time. `https://www.mediego.com/en/`, 2019.

[Mic18] Microsoft. Microsoft smooth streaming (MSS). `https://www.microsoft.com/silverlight/smoothstreaming/`, 2018.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, CHES. Springer, 2017.

[MM09] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, 2009.

[MSO12] Marc Mendonca, Srini Seetharaman, and Katia Obraczka. A flexible in-network ip anonymization service. In *International conference on communications*, ICC. IEEE, 2012.

[Net20] Netflix. Watch tv shows online, watch movies online. `https://www.netflix.com`, 2020.

[NGI] NGINX Inc. NGINX web server. `https://www.nginx.com`.

[NPS04] Alok Nandan, Giovanni Pau, and Paola Salomoni. Ghostshare-reliable and anonymous P2P video distribution. In *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004.*, pages 200–210. IEEE, 2004.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large datasets (how to break anonymity of the netflix prize dataset). In *IEEE Symposium on Security and Privacy*, 2008.

[NSS10] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[OFMR16] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. Confidentiality-preserving publish/subscribe: A survey. *ACM computing surveys (CSUR)*, 49(2):1–43, 2016.

[OTK+18] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 227–240, 2018.

[P2P] P2P Media Loader. Open-source javascript library for p2p media delivery. `https://github.com/novage/p2p-media-loader`.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, Eurocrypt. Springer, 1999.

[Pas12] Andrea Passarella. A survey on content-centric technologies for the current internet: CDN and P2P solutions. *Computer Communications*, 35(1):1–32, 2012.

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

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

[PFC+15] Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latré, and Filip De Turck. QoE-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 12(2):1–24, 2015.

[PGM+18] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. Cyclosa: Decentralizing private web search through SGX-based browser extensions. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 467–477. IEEE, 2018.

[Pli19a] Plista. Artificial intelligence powered recommender as a service. `https://www.recombee.com`, 2019.

[Pli19b] Plista. We turn your content into business outcomes. `https://www.plista.com`, 2019.

[Pro] Project Jupyter. Open-source software, open-standards, and services for interactive computing across dozens of programming languages.

[PSn15] Sutheera Puntheeranurak and Nipith Sa-ngarmangkang. An improvement of video streaming service using dynamic routing over openflow networks. In *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 285–289. IEEE, 2015.

[PVC18] Christian Priebe, Kapil Vaswani, and Manuel Costa. Enclavedb: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 264–278. IEEE, 2018.

[PWC03] Venkata N Padmanabhan, Helen J Wang, and Philip A Chou. Resilient peer-to-peer streaming. In *11th IEEE International Conference on Network Protocols, 2003. Proceedings.*, pages 16–27. IEEE, 2003.

[PZC11] Wei Pu, Zixuan Zou, and Chang Wen Chen. Dynamic adaptive streaming over HTTP from multiple content distribution servers. In *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pages 1–5. IEEE, 2011.

[Qua] Quanteec. Streaming with quanteec: Quality and cost. `https://quanteec.com`.

[RJWS+17] Parisa Rahimzadeh, Carlee Joe-Wong, Kyuyong Shin, Youngbin Im, Jongdeog Lee, and Sangtae Ha. Svc-tchain: Incentivizing good behavior in layered P2P video streaming. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[RVN+16] MA Rajan, Ashley Varghese, N Narendra, Meena Singh, VL Shivraj, Girish Chandra, and P Balamuralidhar. Security and privacy for real time video streaming using hierarchical inner product encryption based publish-subscribe architecture. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 373–380. IEEE, 2016.

[San19] Sandvine. Global internet phenomena report. Technical report, Sandvine, 2019.

[San20] Sandvine. Mobile internet phenomena report. Technical report, Sandvine, 2020.

[SCF+15] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54. IEEE, 2015.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. STD 64, RFC Editor, July 2003. `http://www.rfc-editor.org/rfc/rfc3550.txt`.

[SDK+07] Kyoungwon Suh, Christophe Diot, Jim Kurose, Laurent Massoulie, Christoph Neumann, Don Towsley, and Matteo Varvello. Push-to-peer video-on-demand system: Design and evaluation. *IEEE Journal on Selected Areas in Communications*, 25(9):1706–1716, 2007.

[SES+14] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2014.

[SG16] Jagruti Sahoo and Roch Glitho. Greedy heuristic for replica server placement in cloud based content delivery networks. In *2016 IEEE*

Symposium on Computers and Communication (ISCC), pages 302–309. IEEE, 2016.

[Sha12] Ron Sharp. Latency in cloud-based interactive streaming content. *Bell Labs Technical Journal*, 17(2), 2012.

[SJ14] Yilin Shen and Hongxia Jin. Privacy-preserving personalized recommendation: An instance-based approach via differential privacy. In *International Conference on Data Mining*, ICDE. IEEE, 2014.

[SJWH+17] Kyuyong Shin, Carlee Joe-Wong, Sangtae Ha, Yung Yi, Injong Rhee, and Douglas S Reeves. T-chain: A general incentive scheme for cooperative computing. *IEEE/ACM Transactions on Networking*, 25(4):2122–2137, 2017.

[SKSX18] Hyejin Shin, Sungwook Kim, Junbum Shin, and Xiaokui Xiao. Privacy enhanced matrix factorization for recommendation with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 30(9), 2018.

[SLM+19] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. *arXiv preprint arXiv:1905.05726*, 2019.

[Soc18] Socialx. Socialx whitepaper. `https://socialx.network/wp-content/uploads/2018/02/Whitepaper-SocialX-v0.4.1.compressed-1.pdf`, 2018.

[Sod11] Iraj Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE multimedia*, 18(4):62–67, 2011.

[Sph17] Sphere. Sphere whitepaper. `https://sphere.social/wp-content/uploads/2017/12/Sphere_Whitepaper_v1.7.4.pdf`, 2017.

[SPTH09] Reza Shokri, Pedram Pedarsani, George Theodorakopoulos, and Jean-Pierre Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *3rd ACM conference on Recommender systems*, RecSys. ACM, 2009.

[SRL98] Henning Schulzrinne, Anup Rao, and Robert Lanphier. Real time streaming protocol (RTSP). RFC 2326, RFC Editor, April 1998. http://www.rfc-editor.org/rfc/rfc2326.txt.

[SRR09] Kyuyong Shin, Douglas S Reeves, and Injong Rhee. Treat-before-trick: Free-riding prevention for bittorrent-like peer-to-peer networks. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12. IEEE, 2009.

[Ste18] Steem. Steem whitepaper. https://steem.io/SteemWhitePaper.pdf, 2018.

[Sto18] Storj. Storj whitepaper. https://storj.io/storj.pdf, 2018.

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

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[Ten18] Gil Tene. wrk2: a HTTP benchmarking tool based mostly on wrk. https://github.com/giltene/wrk2, 2018.

[TM12] Bo Tan and Laurent Massoulié. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM transactions on networking*, 21(2):566–579, 2012.

[Twi18] Twinge. Gamesdonequick's streams at twinge. https://twinge.tv/gamesdonequick/streams/#/22233544288, 2018.

[Twi20] Twitch. Twitch is the world's leading video platform and community for gamers. https://www.twitch.tv, 2020.

[VBMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[Vim20] Vimeo. The world's leading professional video platform and community. https://vimeo.com, 2020.

[VPF+18] Sébastien Vaucher, Rafael Pires, Pascal Felber, Marcelo Pasin, Valerio Schiavoni, and Christof Fetzer. SGX-aware container orchestration for heterogeneous clusters. In *38th International Conference on Distributed Computing Systems*, ICDCS. IEEE, 2018.

[VRS+18] Xavier Vilaça, Luís Rodrigues, João Silva, Hugo Miranda, Gustavo Correia, and Tiago Maurício. Fastrank: Practical lightweight tolerance to rational behavior in edge assisted streaming. *Pervasive and Mobile Computing*, 46:18–33, 2018.

[VZ19] André Calero Valdez and Martina Ziefle. The users' perspective on the privacy-utility trade-offs in health recommender systems. *International Journal of Human-Computer Studies*, 121, 2019.

[W3C] W3C. Activitypub decentralized social networking protocol. `https://www.w3.org/TR/activitypub/`.

[WCP+17] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS. ACM, 2017.

[Web] WebRTC. Real-time communication for the web. `https://webrtc.org/`.

[WLM11] Weijie Wu, John CS Lui, and Richard TB Ma. Incentivizing upload capacity in P2P-VoD systems: a game theoretic analysis. In *International Conference on Game Theory for Networks*, pages 337–352. Springer, 2011.

[WML13] Weijie Wu, Richard TB Ma, and John CS Lui. Distributed caching via rewarding: An incentive scheme design in P2P-VoD systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):612–621, 2013.

[WTAR19] Jun Wang, Qiang Tang, Afonso Arriaga, and Peter YA Ryan. Novel collaborative filtering recommender friendly to privacy protection. In *International Joint Conference on Artificial Intelligence*, IJCAI, 2019.

[WXW14] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud. *IEEE/ACM Transactions on Networking*, 23(2):603–615, 2014.

[XLKZ07] Susu Xie, Bo Li, Gabriel Y Keung, and Xinyan Zhang. Coolstreaming: Design, theory, and practice. *IEEE Transactions on multimedia*, 9(8):1661–1671, 2007.

[YLZ+09] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with livesky. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 25–34, 2009.

[You20] YouTube. Share your videos with friends, family, and the world. `https://www.youtube.com`, 2020.

[YZZZ06] Hongliang Yu, Dongdong Zheng, Ben Y Zhao, and Weimin Zheng. Understanding user behavior in large-scale video-on-demand systems. *ACM SIGOPS Operating Systems Review*, 40(4):333–344, 2006.

[ZLHC14] Ge Zhang, Wei Liu, Xiaojun Hei, and Wenqing Cheng. Unreeling xunlei kankan: Understanding hybrid CDN-P2P video-on-demand streaming. *IEEE Transactions on Multimedia*, 17(2):229–242, 2014.

[ZLL15] Shengkai Zhang, Bo Li, and Baochun Li. Presto: Towards fair and efficient HTTP adaptive streaming from multiple servers. In *2015 IEEE international conference on communications (ICC)*, pages 6849–6854. IEEE, 2015.

[ZLM16] Shuai Zhao, Zhu Li, and Deep Medhi. Low delay mpeg dash streaming over the webrtc data channel. In *2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–6. IEEE, 2016.

[ZT15] Hanying Zheng and Xueyan Tang. The server provisioning problem for continuous distributed interactive applications. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):271–285, 2015.

[ZWCR09] Hao Zhang, Jiajun Wang, Minghua Chen, and Kannan Ramchandran. Scaling peer-to-peer video-on-demand systems using helpers. In *2009 16th IEEE international conference on image processing (ICIP)*, pages 3053–3056. IEEE, 2009.

Appendix A

Publications

A.1 PProx

G. Rosinosky, S. Da Silva, S. Ben Mokhtar, D. Négru, L. Réveillère, E. Rivière.
PProx: Efficient Privacy for Recommendation-as-a-Service.
Submitted - 21st International Middleware Conference (Middleware '20).

A.2 PrivaTube

S. Da Silva, S. Ben Mokhtar, S. Contiu, D. Négru, L. Réveillère, E. Rivière.
PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming.
20th International Middleware Conference (Middleware '19).

S. Da Silva.
Priva-Stream: Private Collaborative Live Streaming.
19th International Middleware Conference Doctoral Symposium (Middleware '18).

S. Da Silva.
Priva-Stream: Private Collaborative Streaming.
19th International Middleware Conference Poster (Middleware '18).

A.3 Muslin

S. Da Silva, J. Bruneau-Queyreix, M. Lacaud, D. Négru, L. Réveillère.
MUSLIN: A QoE-Aware CDN Resources Provisioning and Advertising System for Cost-Efficient Multi-Source Live Streaming.
International Journal of Network Management (IJNM '19).

S. Da Silva, J. Bruneau-Queyreix, M. Lacaud, D. Négru, L. Réveillère.
MUSLIN: Achieving High, Fairly Shared QoE Through Multi-Source Live Streaming.
Packet Video Workshop (PV '18).

S. Da Silva, J. Bruneau-Queyreix, M. Lacaud, D. Négru, L. Réveillère.
MUSLIN demo: High QoE Fair Multi-Source Live Streaming.
ACM Multimedia Systems Conference (MMSys '18).

A.4 MS-Stream

J. Bruneau-Queyreix, S. Da Silva, M. Lacaud, D. Négru.
Vers une meilleure diffusion vidéo sur Internet.
Interstices, INRIA, 2018.

A.5 Awards

DASH-IF Excellence in DASH Award
3rd place
MMSys '18

Student Travel Grant Award
MMSys '18

Appendix B

Résumé étendu

B.1 Introduction

Avec l'essor d'Internet pour le grand public sont apparus de nouveaux modes de consommation. Tout d'abord, les contenus vidéo sur des plates-formes telles que YouTube [You20], Vimeo [Vim20] ou Dailymotion [Dai20] augmentent en quantité et en qualité chaque année, notamment grâce à la rémunération des auteurs via la publicité et les partenariats. Il y a également une demande croissante du public pour des contenus spécialisés, dorénavant consommables sur tous supports (ordinateur, tablette, smartphone, téléviseur connecté, etc.). Par ailleurs, les vidéophiles souhaitent visionner des films et séries à leur rythme, sur leur support de prédilection, sans dépendre du programme télévisé. Pour cette raison, des services de vidéo à la demande (VoD) ont émergé. Ces méthodes de diffusion "Over The Top" grâce à des sites web ou des applications deviennent les solutions préférées des consommateurs et consommatrices.

En effet, le streaming vidéo représente actuellement plus de 60% du trafic Internet mondial total [San19], 65% du trafic mobile sur Internet [San20], et devrait atteindre 82% du trafic total d'ici 2022 [Cis18]. Youtube et Netflix représentent à eux seuls plus de 50% du trafic Internet en heure de pointe aux États-Unis. De plus, l'essence même de la télévision (la diffusion en direct) est en train de basculer vers des alternatives sur Internet comme Twitch, privilégiées par le public grâce à leur flexibilité d'utilisation et de support. La part du streaming vidéo en direct sur Internet est également en très forte croissance, puisqu'elle devrait être multipliée par 15 pour atteindre 17% du trafic vidéo total d'ici 2022 [Cis18].

Ces nouveaux modes de consommation de contenus posent un problème majeur : la *qualité d'expérience* proposée à l'utilisateur. En effet, avec une telle croissance, les opérateurs et fournisseurs de contenu peinent à mettre à niveau leurs infrastructures pour supporter la demande toujours croissante des utilisateurs. Les réseaux sont souvent saturés, les serveurs se retrouvent surchargés, et il devient de plus en plus difficile de proposer une diffusion fiable, sans coupures, avec une bonne qualité visuelle et une stabilité satisfaisante, à des coûts abordables pour les fournisseurs de contenus. Il est alors nécessaire de trouver des solutions pour réduire l'impact de ces flux sur la santé du réseau, en permettant au plus grand nombre d'accéder aux ressources tout en fournissant une bonne qualité d'expérience aux utilisateurs consommant les contenus.

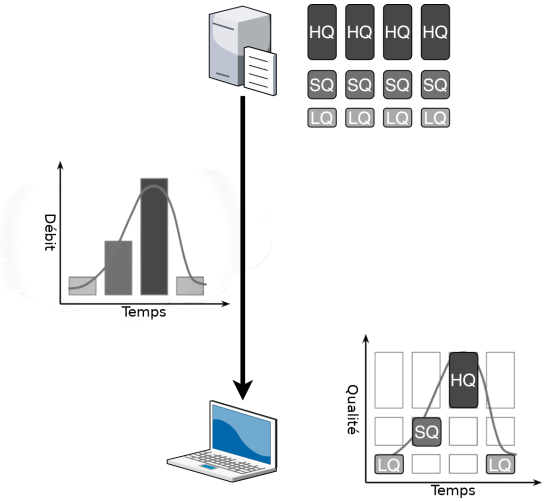


Figure B.1: Streaming adaptatif sur HTTP

Des méthodes de streaming vidéo adaptatif sur HTTP ont récemment vu le jour, telles que Adobe HDS [Ado18], Apple HLS [App18], Microsoft SS [Mic18], et notamment le standard DASH [Sod11] qui est utilisé entre autres par YouTube, Facebook, Netflix et Twitch. L'objectif de ces techniques est de réduire le nombre de coupures lors de la diffusion de vidéos en adaptant la qualité du flux à la bande passante disponible entre l'utilisateur et le serveur (voir Figure B.1). Pour cela, la vidéo est d'abord encodée dans plusieurs qualités. Ensuite, chaque qualité est découpée en segments de quelques secondes. Quand le client souhaite recevoir une vidéo, le serveur lui fournit une liste des différentes qualités disponibles, et le lecteur vidéo choisit alors la qualité la plus adaptée à la bande passante disponible pour chaque segment. La vidéo est alors reçue en plusieurs segments qu'il faut remettre bout à bout pour lire le flux.

B.2 Motivation

Le standard DASH et les techniques similaires permettent d'améliorer sensiblement la *qualité d'expérience* du public en éliminant la plupart des coupures dues aux mauvaises conditions du réseau entre l'utilisateur et le serveur. En revanche, les problèmes liés à la surcharge des serveurs ou à leur capacité perdurent. Si de nombreux utilisateurs situés dans la même zone géographique regardent simultanément un même contenu vidéo, le serveur le plus proche devient rapidement surchargé (voir Figure B.2). Certains utilisateurs subissent alors des dégradations de qualité ou une indisponibilité du contenu, et donc une *qualité d'expérience* faible et inéquitable.

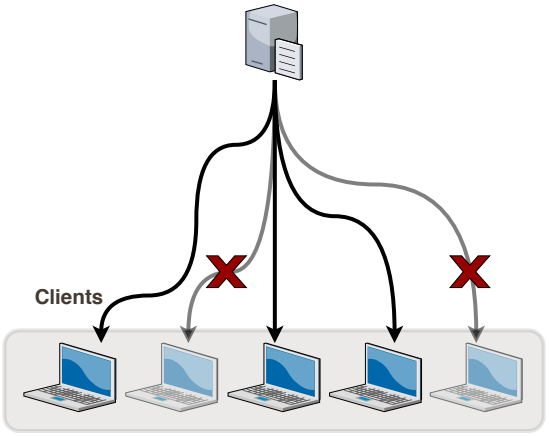


Figure B.2: Congestion d'un serveur DASH

Une autre problématique importante est la protection de la *vie privée*. L'utilisation des plateformes de streaming génère des informations personnelles sensibles en terme d'historique de visionnage aux vidéos. Ces données peuvent être exploitées soit au bénéfice de l'utilisateur, par exemple pour lui faire des recommandations personnalisées pour d'autres contenus, ou au bénéfice de la plateforme pour de la publicité ciblée. Cependant, la disponibilité des historiques d'accès peut également conduire à des menaces majeures sur la vie privée. En effet, il est facilement possible d'inférer des informations privées sur l'utilisateur, tel que son genre, origine, ses orientations politiques, religieuses ou sexuelles [KMSG13], ou la composition du domicile familial [LXZ+14].

Objectif

L'objectif de cette thèse est de proposer un système pragmatique et réaliste de streaming vidéo préservant la vie privée, proposant à la fois une meilleure *qualité d'expérience* et des garanties de protection de la *vie privée* aux utilisateurs, au moindre coût. Proposer une bonne *qualité d'expérience* signifie (1) fournir une qualité d'image haute, (2) minimiser les fluctuations de qualité, (3) éviter les interruptions pendant la lecture, et (4) assurer un temps de démarrage rapide [SES+14]. Protéger la *vie privée* des utilisateurs dans un système de streaming vidéo signifie camoufler leur historique de visionnage, à la fois des serveurs et des autres utilisateurs.

B.3 Contexte

Les solutions utilisant le standard DASH sont très efficaces pour faire face aux fluctuations de bande passante entre l'utilisateur et le serveur. En revanche, lorsque certains serveurs sont surchargés ou que des liens réseau sont saturés, il est judicieux de récupérer simultanément le contenu vidéo depuis plusieurs sources différentes afin de maximiser la qualité d'expérience de l'utilisateur [AGH+12]. Plusieurs protocoles de streaming vidéo multi-sources ont récemment émergé pour faire face à la surcharge des serveurs ou liens réseau [ZLL15, PZC11].

MS-Stream

MS-Stream (Multiple-Source Streaming) [BQLN+18, BQLN+17b, BQLN+17c], conçu au LaBRI, est un protocole de streaming vidéo adaptatif compatible avec DASH. Il permet d'utiliser plusieurs serveurs simultanément pour assurer une meilleure *qualité d'expérience* au public, à la fois en réduisant le nombre de coupures et en améliorant la qualité vidéo affichée (grâce à l'agrégation des bandes passantes).

Tout comme pour DASH, la vidéo est d'abord encodée en différentes qualités puis découpée en segments contenant plusieurs groupes d'images. Le contenu est ensuite copié sur plusieurs serveurs différents. Lorsque l'utilisateur souhaite regarder un segment d'un contenu vidéo, le lecteur vidéo fait des requêtes auprès des différents serveurs disponibles). Chaque serveur va alors proposer un sous-segment composé de groupes d'images en bonne qualité et d'autres en qualité basse, en fonction de la bande passante disponible et de

sa capacité. De cette manière, le client peut rassembler les différents groupes d'images reçus pour reformer un segment en bonne qualité. Si certains groupes d'images en bonne qualité ne sont pas reçus, il est possible d'utiliser ceux de basse qualité fournis par les autres serveurs afin de compléter le segment et continuer la lecture sans interruption.

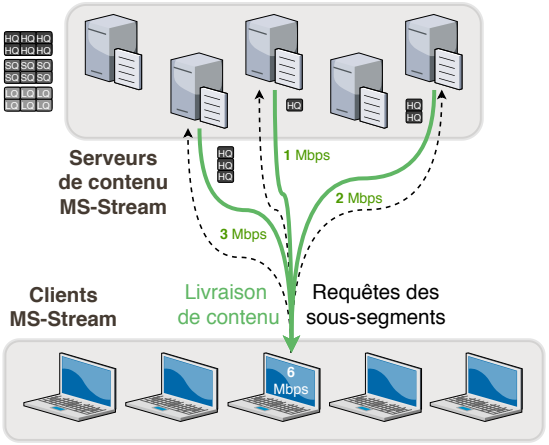


Figure B.3: Aggrégation de bande passante avec MS-Stream

Dans l'exemple de la Figure B.3, l'utilisateur a une bande passante de 3 Mbps avec un serveur, 1 Mbps avec le deuxième, et 2 Mbps avec le troisième, une qualité visuelle allant jusqu'à 6 Mbps pourra être obtenue. Cette qualité est alors supérieure à la qualité que DASH aurait pu fournir (ici au maximum 3 Mbps avec le premier serveur). Si maintenant le deuxième serveur devient surchargé ou indisponible, l'utilisateur pourra toujours obtenir 3+2 = 5 Mbps depuis les deux autres serveurs, sans que cela n'interrompe la lecture.

Dans MS-Stream, le client utilise donc les groupes d'images redondants de basse qualité dans l'éventualité où ceux en bonne qualité ne sont pas reçus a temps. La surcharge subie par le réseau en bande passante dépend alors de la qualité des vidéos. En moyenne, nous observons moins de 10% d'augmentation de la bande passante utilisée lors de nos évaluations. De plus, la génération et agrégation des sous-segments a une empreinte minime [BQLN+17c] puisqu'il suffit d'assembler des groupes d'images déjà encodés en différentes qualités par ailleurs.

B.4 `Muslin`

Les services de streaming dépendent de larges réseaux de serveurs pour héberger le contenu vidéo. Les utilisateurs sont automatiquement redirigés vers le serveur le plus proche d'eux afin de mitiger les saturations et atteindre un meilleur débit. Cependant, si un large nombre d'utilisateurs situés dans la même région visionnent simultanément un flux vidéo, le serveur le plus proche peut rapidement être surchargé.

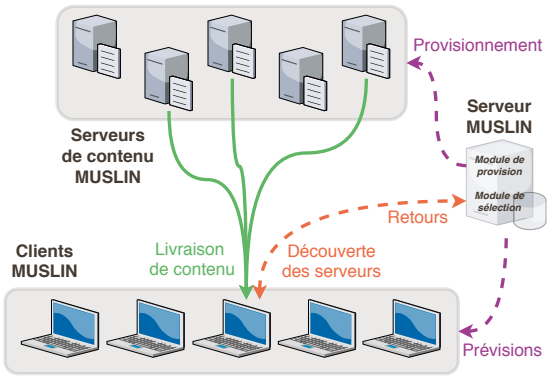
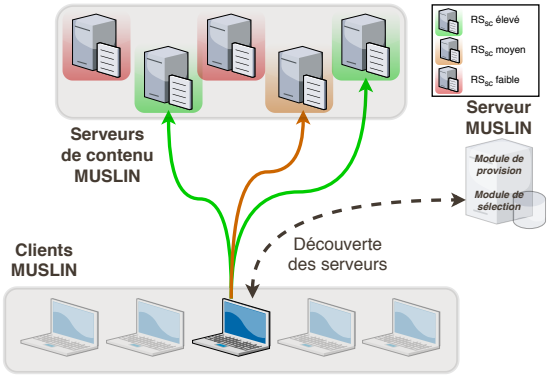


Figure B.4: Vue d'ensemble de `Muslin`

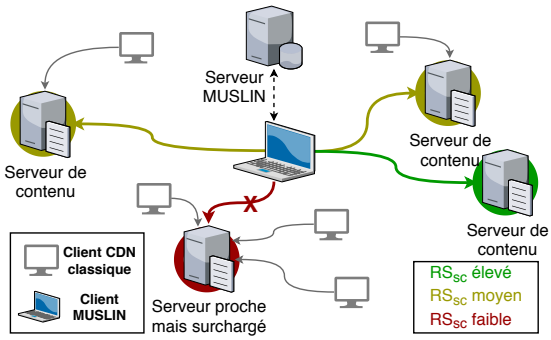
`Muslin` [DSBQL$^+$19, DSBQL$^+$18a, DSBQL$^+$18b] est une solution de streaming vidéo fournissant une *qualité d'expérience* haute et équitable aux utilisateurs, nécessitant une infrastructure moindre que les solutions actuelles. `Muslin` implémente MS-Stream pour la livraison du contenu afin d'agréger les bandes passantes. `Muslin` utilise des retours périodiques automatisés des lecteurs vidéo des clients pendant les sessions de streaming ainsi qu'un score de classement pour provisionner et affecter dynamiquement les serveurs selon de multiples critères. Cela permet d'ajuster l'échelle de l'infrastructure en temps réel en fonction du besoin constaté et donc réduire les coûts.

Comme montré sur la Figure B.4, le module de provisionnement ajuste dynamiquement le nombre de serveurs en fonction des besoins constatés et de l'estimation de la bande passante nécessaire.

Le module de sélection affecte des serveurs aux clients en fonction de plusieurs critères, tels que la distance, bande passante et charge, aggrégés dans un score de classement RS_{sc} (voir Figure B.5a). Comme illustré sur la Figure B.5b, le serveur le plus proche n'est pas toujours le plus pertinent à affecter aux utilisateurs.



(a) Exemple de la sélection des serveurs grâce au RS_{sc} calculé par `Muslin`



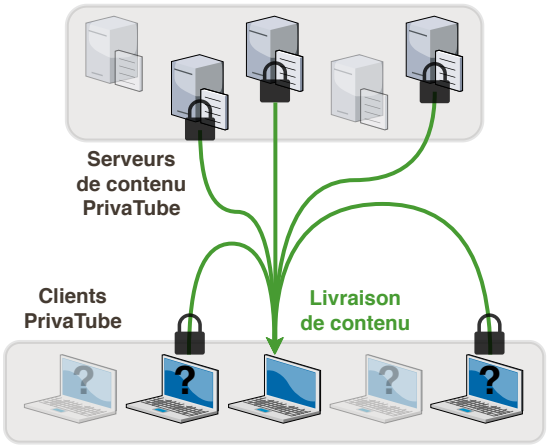
(b) Pertinence du RS_{sc} de `Muslin`

Nous avons utilisé `Muslin` pour rejouer un flux d'une journée couvrant un événement de jeux vidéos, avec plusieurs centaines de clients et en testant des configurations différentes. Les résultats montrent que notre approche surpasse les méthodes classiques en améliorant la *qualité d'expérience* et l'équité entre utilisateurs (suppression totale des coupures, amélioration de la qualité vidéo et diminution des fluctuations), tout en nécessitant moins de serveurs (environ -18%).

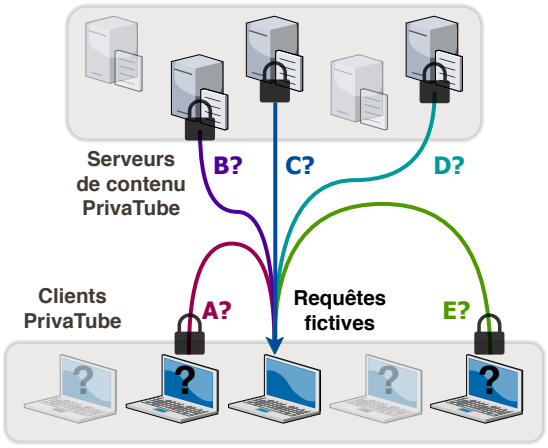
B.5 PrivaTube

Transmettre des flux vidéo de manière fiable et à large échelle requiert une grande plateforme de diffusion avec de nombreux serveurs. Cependant, les historiques d'accès peuvent révéler des informations sensibles, et les plateformes d'hébergement sont connues pour exploiter les données personnelles. Il est donc nécessaire de protéger les intérêts des utilisateurs pour concevoir une nouvelle génération de services de streaming.

PrivaTube [DSBMC+19] est un système de streaming vidéo pragmatique fournissant une bonne *qualité d'expérience* à ses utilisateurs tout en protégeant leur *vie privée*. PrivaTube étend MS-Stream pour améliorer la *qualité d'expérience*, réduire la charge sur les serveurs et le coût de l'infrastructure en permettant aux clients de récupérer les contenus à la fois depuis les serveurs centraux et depuis les pairs ayant regardé le même contenu précédemment (voir Figure B.6a). PrivaTube protège la *vie privée* des utilisateurs en chiffrant tous les flux dans des environnements d'exécution de confiance Intel SGX, à la fois côté client et serveur (voir Figure B.7). De plus, des requêtes fictives permettent de brouiller les pistes (voir Figure B.6b).



(a) Vue d'ensemble de PrivaTube



(b) Requêtes fictives de PrivaTube

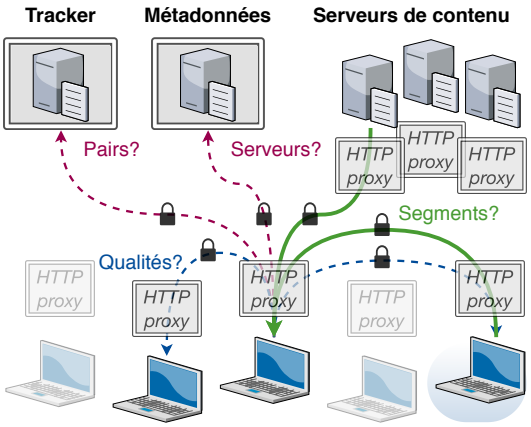


Figure B.7: Architecture de PrivaTube

En effet, PrivaTube permet de paramétrer la découverte probabiliste du lien entre un utilisateur et une vidéo à hauteur d'un pourcentage δ. Ces requêtes fictives ont un surcoût qui est exploité par le système en pré-provisionnant du contenu chez les pairs afin d'améliorer la disponibilité et le passage à l'échelle. Cela permet ainsi d'améliorer la *qualité d'expérience* grâce à l'agrégation des bandes passantes depuis plusieurs sources, notamment pour les vidéos moins populaires.

Nous avons implémenté PrivaTube et l'avons déployé sur un réseau de 14 machines pour évaluer ses performances et son comportement. Nous avons également conduit des simulations à grande échelle sur des jeux de données réels d'historiques d'accès à des vidéos. Nos résultats démontrent que PrivaTube offre un anonymat quasi-total aux utilisateurs tout en proposant une meilleure *qualité d'expérience* que les systèmes actuels.

La durée de téléchargements des segments est 2 à 15 fois plus rapide, la qualité vidéo entre 10% et 300% plus élevée, pour un surcoût de charge serveur de 17% et un délai de démarrage supplémentaire de seulement 40ms.

B.6 PProx

Les plateformes de streaming vidéo (telles que YouTube [You20], Vimeo [Vim20] ou Dailymotion [Dai20]), proposent des recommandations de contenus aux utilisateurs afin de les conserver sur leur site ou application. Pour cela, elles peuvent soit établir des profils d'intérêts pour les utilisateurs, soit dépendre de services de recommandations externes. Le calcul de ces recommandations est toujours basé sur l'historique de navigation, et parfois sur des données entrées par les utilisateurs. Cela pose donc des menaces à la *vie privée*, puisque (i) les fournisseurs de service collectent des données personnelles, (ii) un attaquant peut intercepter les recommandations et déduire des informations privées sur l'utilisateur, et (iii) des plateformes malveillantes peuvent cibler des utilisateurs spécifiques avec de la publicité pour générer des revenus, au lieu de les segmenter par groupes d'intérêts.

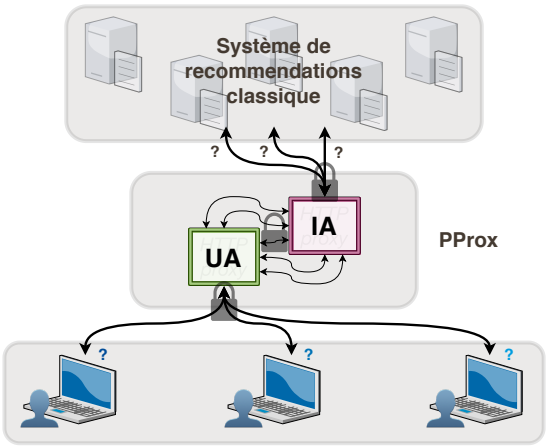
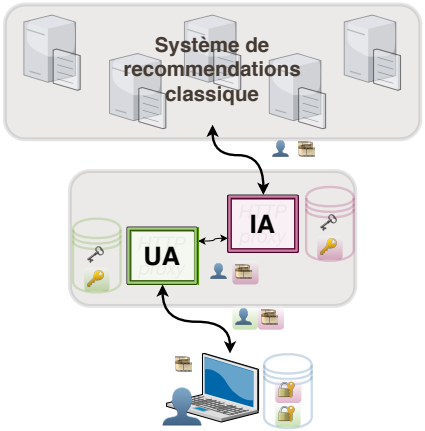


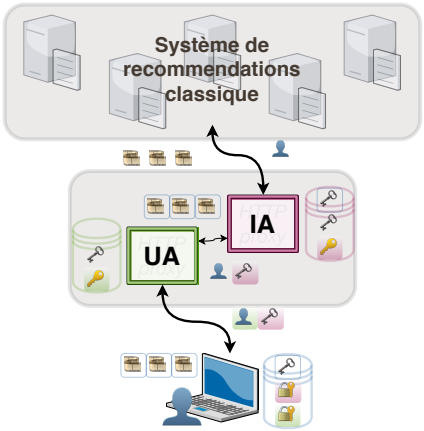
Figure B.8: Vue d'ensemble de PProx

PProx est une solution pragmatique permettant de fournir un service de recommandations aux utilisateurs des plateformes de streaming tout en préservant leur *vie privée*, en garantissant un anonymat total. PProx permet une bonne *qualité d'expérience* puisqu'il n'impacte pas la précision ou la nature des recommandations, et peut être déployé avec des contraintes minimales. Il dépend d'un système de double proxy dans des enclaves Intel

SGX, situé entre l'utilisateur et le service de recommandations, qui chiffre et anonymise les requêtes à la volée de manière transparente. Il mélange également les requêtes des différents clients afin de casser définitivement le lien entre les utilisateurs et les contenus qu'ils visionnent ou reçoivent comme recommandation (voir Figure B.8). Ce principe est robuste aux attaques de type *side-channel*, et même à la compromission d'une des enclaves. `PProx` passe à l'échelle de manière élastique et dynamique sur un réseau de machines disposant d'enclaves Intel SGX.



(a) `PProx` - Insertion d'un élément



(b) `PProx` - Envoi des recommendations

Nous avons connecté `PProx` avec le système de recommandations intégré dans Harness et l'avons évalué sur un cluster de 27 machines. Les résultats démontrent la capacité de `PProx` à gérer un grand nombre de requêtes avec une faible latence (moins de 100ms contre plusieurs secondes pour les systèmes similaires actuels), permettant d'atteindre la charge maximale supportée par le système de recommandations avec un surcoût acceptable (seulement 30% à 50% de nœuds en plus).

B.7 Conclusion

Le streaming vidéo évolue très rapidement et rencontre de nombreux défis techniques et technologiques. Nous croyons que notre travail prouve que sécurité et protection de la vie privée des utilisateurs ne rime plus avec faibles performances et basse qualité d'expérience. De nombreuses solutions pragmatiques peuvent être développées dans l'industrie en se basant sur nos contributions.

Nous espérons que généraliser `Muslin`, PrivaTube et `PProx` pourra permettre à une nouvelle génération de services de streaming vidéo respectant la vie privée de voir le jour.