UNIVERSIDAD DE ALMERÍA

EFFICIENCY AND SECURITY IN PEER-TO-PEER STREAMING PROTOCOLS

Author
CRISTÓBAL MEDINA LÓPEZ
Supervisors
Dr. LEOCADIO GONZÁLEZ CASADO
Dr. VICENTE GONZÁLEZ RUIZ

Department of Informatics
Almería, 2019

UNIVERSIDAD DE ALMERÍA

EFICIENCIA Y SEGURIDAD EN PROTOCOLOS DE STREAMING PEER-TO-PEER

Autor
CRISTÓBAL MEDINA LÓPEZ

Directores
Dr. LEOCADIO GONZÁLEZ CASADO
Dr. VICENTE GONZÁLEZ RUIZ

Department of Informatics
Almería, 2019

Título en Inglés: *Efficiency and Security in Peer-to-Peer Streaming Protocols.*
Título en Español: *Eficiencia y Seguridad en Protocolos de Streaming Peer-to-Peer.*
Programa de doctorado: Doctorado en Informática (RD99/11).
Doctorando: Cristóbal Medina López.
Directores: Leocadio González Casado y Vicente González Ruiz.

La siguiente página web contiene información actualizada sobre los temas relacionados con esta tesis doctoral:
https://hpca.ual.es/
Texto impreso en Almería, septiembre de 2019

First, solve the problem. Then, write the code.

- John Johnson

Preface

Internet has changed the way we communicate. The use of mobile devices together with the increase in the bandwidth has favoured the popularity of video streaming services. For example, video games streaming and video-conferencing are used massively nowadays. This fact has led the appearance of a multitude of streaming architectures, which basically, consist in transmitting one or several multimedia signals with as much quality as possible to as many users as necessary in the shorter possible time.

There exist many systems broadcasting the same content to many users. Nowadays, the most used solution for live multimedia streaming is the Client/Server model. It is based on the idea that the stream is generated in only one point (server), on-demand (pull model) or simply under data availability (push model), and it is transmitted to a set of clients. However, by the very nature of the model, pure C/S systems only scale with the number of users if replication is used in the servers side. Because of that, most of the C/S streaming platform use CDNs (Content Delivery Networks), which is a set of servers with the content duplicated which increases the cost of the infrastructure. On the other hand, in the P2P (Peer-to-Peer) model only one copy of the stream is sent by the server, and the peers (clients) are in charge of retransmitting the stream as many times as necessary (usually the number of peers) by using their upload bandwidth. This means an important cost-saving because most of the resources are provided by the peers. However, there exist some issues causing that P2P currently not be the facto standard:

- Security. The anonymity provided by most P2P networks leads to malicious peers attacking in different ways: denial of service attacks, poisoning of content, avoiding sharing their resources, etc.

- Latency. Because of its distributed approach, some times there is a considerable delay from when the content is generated until users play it.
- Limitations in networks. Mainly caused by peers behind NATs and firewalls.
- Resources needed on the client side. The complexity of most of the P2P applications, result in they can not be executed on some types of device and/or networks. Especially in those with low resources.

This thesis is focused on resolve previous issues.

Regarding security, several strategies to face pollution attacks are proposed and analysed. We focus on persistent attacks, selective attacks, and collaborative attacks. Among the contributions of this thesis there are non-deterministic techniques such as the incorporation of trusted peers into the P2P overlay (team), the use of digital signatures, and the application of Shamir secret sharing to enforce peers to collaborate in order to remain in the team. Moreover, some novel techniques are also proposed, mainly applied to Software Defined Networks (SDNs). The goal is to detect and expel malicious peers without producing false positive or false negative results. This approach allows a quick detection, not only in pure SDN environment but also in a mixed environment where some peers are in SDN but others are on the Internet.

Latency is another addressed issue. Most P2P protocols are specifically designed for real time streaming. The main objective is to keep it as straightforward as possible reducing the communication cost to the maximum. To accomplish with such requirement, our proposal, P2P Straightforward Protocol (P2PSP), use a push-based model under a fully-connected mesh topology. Peers connect directly with their neighbors and video chunks are sent without a previous request. A novel control congestion mechanism based on the idea that "Only if I have received a chunk, I should send a chunk" has been included in P2PSP.

Communications can be also an important issue because NATs and firewalls usually produce communication problems between two peers. Techniques to traverse NATs are proposed. The main objective is to increase the chances two peers perform a successful communication, increasing in that way the efficiency.

Another problem is the limited use of P2P protocols on the client side, due mainly to high consumption of resources. A proposal has been developed and analysed, which allows the implementation of a light version of the P2PSP protocol which is able to be executed in any current Web browser, even in low-cost and low-resource devices such as the Google Chromecast.

Prólogo

Internet ha cambiado la forma en la que nos comunicamos. El uso de dispositivos móviles unido al crecimiento generalizado en el ancho de banda de las conexiones ha propiciado un incremento considerable del uso de servicios de streaming de vídeo, como por ejemplo, en la emisión de las partidas de videojuegos online o en las videoconferencias. Esto ha motivado que aparezcan multitud de arquitecturas de streaming que, en esencia, tratan de transmitir una o varias señales multimedia con tanta calidad como sea posible a tantos usuarios como sea necesario.

El interés de esta tesis se centra, principalmente, en aquellos sistemas que difunden un único contenido a muchos usuarios. La solución para streaming de vídeo en directo más usada hoy en día es el modelo Cliente/Servidor. Dicha propuesta se basa en la idea de que el stream es generado en un único punto (el servidor), bajo demanda (pull model) o simplemente por disponibilidad de los datos (push model), y es transmitido hasta un conjunto de clientes. Sin embargo, ha de tenerse en cuenta que, por la propia naturaleza del modelo, los sistemas puros C/S escalan mal con el número de usuarios si no se usa replicación en la parte servidora. Por este motivo, la mayoría de los portales de streaming del tipo C/S utilizan CDNs (Content Delivery Networks), que en definitiva no son más que un conjunto de servidores con el contenido replicado, aumentando así los costes de infraestructura.

Por otro lado, también se dispone del modelo P2P (Peer-to-Peer). En éste, la parte servidora únicamente envía una o unas pocas copias del stream y son los peers (los clientes) los que usan su ancho de banda de subida excedente para replicar el stream tantas veces como sea necesario, generalmente, tantas como peers. Esto supone un ahorro de costes considerable ya que los recursos los proporcionan los peers. No obstante, existen algunos problemas que hacen que las redes P2P no sean el estándar de facto para el streaming de contenidos multimedia:

- Seguridad. El anonimato que proporciona la mayoría de estas redes da lugar a que peers maliciosos formen parte de la misma y la ataquen de diferentes modos: ataques de denegación de servicio, envenenamiento del contenido, espionaje entre peers, evitando compartir sus recursos, etc.
- Latencia. Debido a su carácter distribuido, en ocasiones se provoca un retardo considerable desde que se genera el contenido hasta que los usuarios lo consumen.
- Limitaciones en las comunicaciones. Principalmente propiciadas por el uso de NATs y cortafuegos en la red.
- Recursos necesarios en el cliente. La complejidad de algunas aplicaciones P2P hace que no puedan ser ejecutadas en cualquier tipo de dispositivo y/o red. Especialmente en aquellos que disponen de bajos recursos.

El desarrollo de esta tesis doctoral se centra en resolver estos problemas.

En cuanto a la seguridad, se proponen y analizan diferentes estrategias centradas en resolver los ataques de modificación del contenido (pollution attacks) en sus variantes principales: ataques persistentes, selectivos, y colaborativos. Entre las contribuciones se encuentran técnicas no deterministas como la introducción de peer monitores *trusted peers*, el uso de firmas digitales, y la aplicación del algoritmo de compartición de secretos de *Shamir* para forzar a los peers a colaborar si quieren seguir formando parte de la transmisión. También se pueden encontrar otras técnicas deterministas muy novedosas, aplicadas principalmente a las redes definidas por software (*Software Defined Networks*, SDN) que permiten detectar y expulsar al atacante sin generar falsos positivos o falsos negativos. Esto permite una detección rápida, no solo en entornos puramente SDN, sino también en entornos mixtos donde algunos peers están en redes SDN y otros, por ejemplo, en Internet.

Respecto a la latencia, esta tesis propone un protocolo P2P diseñado específicamente para el streaming de contenido manteniendolo lo más simple posible, tratando de reducir al máximo los costes de las comunicaciones. Es por ello que el protocolo P2PSP se basa en un modelo *push-based* con una topología en malla completamente conectada, donde la comunicación se establece entre pares vecinos y los chunks se envían sin ser explícitamente solicitados. Cabe destacar el sistema de control de la congestión, que se basa en una idea básica: “Sólo cuando se recibe un mensaje, se debe enviar un mensaje”.

La comunicaciones también pueden ser una gran problema, ya que el uso de NATs y/o cortafuegos pueden impedir que dos peer de una red P2P puedan comunicarse con normalidad. Es por ello que se han propuesto y analizado técnicas para atraversar NATs, aumentando la probabilidad de conseguir conexiones exitosas entre pares, y mejorando por tanto su eficiencia.

Otro de los aspectos destacados entre los problemas de las aplicaciones P2P es su uso limitado en la parte del cliente, debido principalmente a un alto consumo de recursos para un correcto funcionamiento. En este sentido, se ha desarrollado y analizado una propuesta, junto con su correspondiente implementación del protocolo P2PSP, que permite materializar una versión liviana capaz de ser ejecutada en cualquier navegador Web moderno, incluso en dispositivos de bajo coste y bajos recursos como el Google Chromecast.

Agradecimientos

A mis padres, por darme la oportunidad de estar escribiendo estas líneas. Por confiar siempre en mí y enseñarme el verdadero valor de las cosas.

A Macarena, por estar siempre conmigo, por escucharme y ayudarme a tomar el mejor camino en las decisiones difíciles. Porque a su lado todo es más fácil.

A mis hermanos, familia y amigos. Por estar ahí, y compartir los mejores momentos (y los no tanto) conmigo.

A Vicente y Leocadio, directores de este trabajo. Por su entrega, ayuda incondicional y por darme la oportunidad de aprender a su lado. Por escuchar mis ideas, discutirlas y valorarlas. Por todo lo que me han enseñado y por las experiencias que hemos vivido juntos estos años.

A Yuansong Qiao y Brian Lee, por acogerme durante mi estancia de tres meses en el Software Research Institute (SRI) del Athlone Institute of Technology en Irlanda. A todos los miembros del SRI por hacerme sentir uno más, y por las interesantes charlas en los cafés.

A todo el grupo Supercomputación-Algoritmos (TIC 146) de la Universidad de Almería, y en especial, a los que compartimos pasillo en el CITIC. Han sido unos años fantásticos en los que hemos compartido muchos cafés, charlas y debates muy interesantes (académicos o no).

En general, a todos los que se han cruzado en mi camino, hemos colaborado o compartido experiencias.

Por último, agradecer también al Ministerio de Educación, Cultura y Deporte, por su financiación para realizar el doctorado durante estos años a través de una ayuda para Formación del Profesorado Universitario (FPU) con código FPU14/00635. Así como por la financiación recibida para realizar la estancia de 3 meses en el Athlone Institute of Technology (Irlanda).

A todos, ¡muchas gracias!

Septiembre de 2019

Acronyms

ACS	Adaptive Capacity Set.
ALM	Application Layer Multicast.
API	Application Programming Interface.
CDN	Content Delivery Network.
CIS	Content Integrity Set.
CLR	Chunk Loss Rate.
CPU	Central Processing Unit.
CS	Client-Server.
DBS	Data Broadcasting Set.
DoS	Deny of Service.
DTLS	Datagram Transport Layer Security.
DVB	Digital Video Broadcasting.
DYNAT	Dynamic Network Address Translation.
ESCLR	Exponential Smoothing of Chunks Loss Rate.
FCS	Free-riding Control Set.
GC	Google Chromecast.
HD	High Definition.
HDMI	High-Definition Multimedia Interface.
HEVC	High Efficiency Video Coding.
HP	Honest Peer.
HTML	HyperTex Markup Language.
HTTP	Hypertext Transfer Protocol.
ICE	Interactive Connectivity Establishment.
IETF	Internet Engineering Task Force.
IMS	IP Multicast Set.
IP	Internet Protocol.
IPM	IP Multicast.
IPTV	Internet Protocol Television.
ISP	Internet Service Provider.
JSON	JavaScript Object Notation.
LAN	Local Area Network.
LBS	Load Balancing Set.
MCS	Multi-Channel Set.
MDC	Multiple Description Coding.
MP	Malicious Peer.
MRS	Massively-lost chunk Recovery Set.
MSE	Media Source Extensions.
MTD	Moving Target Defense.
NAT	Network Address Traslation.
NAY	Not Attacked Yet.
NTS	NAT Traversal Set.
OF	OpenFlow.
OF-RHM	OpenFlow Random Host Mutation.
P2P	Peer-to-Peer.
P2PSP	Peer-to-Peer Straightforward Protocol.
PC	Personal Computer.
QoE	Quality of Experience.
RAM	Random Access Memory.
RFC	Request for Comments.

ACRONYMS

RSA	Rivest–Shamir–Adleman (public-key cryptosystems).
SDN	Software Defined Networks.
SDP	Session Description Protocol.
SIP	Session Initiation Protocol.
SSS	Shamir's Secret Sharing.
STCP	Scalable Transmission Control Protocol.
STrPe	Strategy based on Trusted Peers.
STrPe-DS	Strategy based on Trusted Peers and Digital Signatures.
STUN	Session Traversal Utilities for NAT.
SVC	Scalable Video Coding.
TAS	Topology Adaptation Set.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TP	Trusted Peer.
TTP	Trusted Third Party.
TURN	Traversal Using Relays around NAT.
UDP	User Datagram Protocol.
URL	Universal Resource Locator.
VLAN	Virtual LAN.
VoD	Video on Demand.
W3C	World Wide Web Consortium.
WebRTC	Web Real Time Communication.
XMPP	Extensible Messaging and Presence Protocol.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter
1

Introduction

Massive distribution of real-time video content is one of the big challenges on the Internet. Nowadays, there are several proposals that approximate this goal, but none of them provide a QoS (Quality of Service) comparable to the DVB (Digital Video Broadcasting), for example. This is a direct consequence of the design of the Internet and an inefficient use of the capacity of the network. The Internet was created to provide the so called best-effort service that means that you can transmit data through the network but the transmission time is unknown a priori. That time depends on several factors such as network failures, the network load and, of course, the amount of sent data. These two last factors are directly related to the capacity of the network, a term also referred as network bandwidth.

In order to provide an acceptable QoS in real-time streaming scenarios, one of the most important requirements to fulfill is to have a lower bound of the network capacity. However, most of the current solutions fail to take advantage of this resource. One of the reasons is that IP multicast has not achieved the expected popularity. This forces content sources to replicate the same data for different receivers, resulting in linear growth of the transmitted data when the number of receivers increases. In this situation, P2P (Peer-to-Peer) overlays can improve the performance of real-time streaming services: given that all peers manage (and thus can share) the same content at the same time, the transmission requirements at the source are dramatically reduced.

This work introduces P2PSP (Peer-to-Peer Straightforward Protocol) in Chapter 2. P2PSP is a set of transmission and machine behavior rules that helps to increase the QoS of real-time streaming systems. Basically, the P2PSP mimics the IP multicast solution by taking advantage of the transmission capacity of the peers which is typically wasted in pure C/S (Client/Server) services. We would like to stress that, although the P2PSP can be a standalone solution for small scenarios, its performance can be increased in massive scenarios by taking advantage of both paradigms: the C/S model and the P2P model.

However, it is important to take into account that P2P networks are vulnerable. Pollution attacks are one of the most important problems to face with. In order to mitigate those attacks, a set of strategies for traditional networks have been proposed in Chapter 3. Some of them based on Trusted Peers (TPs) and the use of digital signatures, and another with the aim of enforcing peers to collaborate by using an approach based on Shamir's Secret Sharing scheme. Unfortunately, the use of TPs as the main defense strategy is not appealing when the number of malicious is large. Moreover, the techniques based on Shamir do not resolve all possible types of attacks. Then, we require an adaptive algorithm which some times can not be applied in all scenarios. These drawbacks led us to face this problem from a more novel network management concept: Software Defined Networks (SDNs).

SDN aims to facilitate network management as well as enable efficient programming of network configuration. SDN networks are interesting for us to face the static nature of traditional network infrastructures. Taking into account the potential advantages provided for the SDN technology, a method for detecting and expelling malicious peers in a deterministic way is proposed in Chapter 4. This technique could avoid the false positive and false negative results observed in other methods such as peer polling or trust management systems. The experiments demonstrate the viability of the proposal, resulting in a relatively quick MP detection (concerning the size of the overlay), not only in pure SDN environments but also in mixed environments where some peers are on the Internet and others are under managed networks.

However, despite pollution attacks could be successfully mitigated, others issues regarding the communication between peers may appear. For instance, the use of NAT devices, which hinder the communication between processes running in the internal and external networks. Some protocols, such as STUN and ICE, have been developed to cope with this problem, but they are not enough for traversing all existing NAT types. For that reason, a new and simple NAT Traversal algorithm that uses Collaborative Port Prediction (NT-CPP) is proposed in Chapter 5. It provides an effective NAT traversal for all combinations of port assignment and endpoints filtering strategies, as long as the NAT behaviour can be predicted.

Finishing, one of the most interesting advantages of the P2PSP is that it is suitable for broadcasting real-time events among low-end devices. Today, a large part of client devices are mobile. However, their small screens provide poor QoE, especially for high-definition contents. For that reason, having in mind devices with limited resources, we integrate several Web technologies into the development of a P2P system for Google Chromecast in Chapter 6. Our experiments show that our proposal works smoothly on resource constrained devices, without third-party media servers, which increases privacy while reducing costs.

Chapter

2

Peer to Peer Straightforward Protocol (P2PSP)

Peer to Peer Straightforward Protocol (P2PSP) is an application-layer protocol that provides real-time broadcasting, also known as Application Layer Multicast (ALM), of a media stream on the Internet. Peers collaborate to disseminate the stream that is generated by a single source, focusing on minimizing the latency and the protocol overhead. P2PSP overlays are push-based (topology-driven) meshes. P2PSP establishes communication between all peers, and chunks of data are flooded without explicit requests. P2PSP has a modular design organized in *sets of rules*, where each module is specialized in implementing different functionalities.

2.1 Related Work on P2P Live Streaming Systems

In the last two decades, there has been a vast research effort in the field of massive streaming of multimedia content, and particularly in the case of live-media by using P2P overlay networks. In this context, a single source generates the content which must be broadcasted to the receivers (the end-users) as soon as possible (minimizing the latency) and without a significant loss of information. Notice that this functionality can be obtained with IPM (IP Multicast) [18]. Unfortunately, the service model provided by IPM does not fit the needs of most of the content providers. Moreover, it generates some problems for ISPs (Internet Service Providers), such as security and bandwidth consumption concerns [20, 27]. Consequently, IPM is not globally available, at least for the end-users.

DONeT (Coolstreaming) [76] is one of the first and most famous ALM proposals, which have attracted millions of users. DONeT is a pure *pull-based random-mesh P2P overlay* for the streaming of live-media from a single source, inspired by the ideas of BitTorrent [10]. As in BitTorrent, each DONeT peer tries to establish a number of connections with other peers, retrieves their buffer maps[1] from them, decides what chunks will be requested to each peer, in which order, and finally explicitly retrieves the chunks. Because the order in which the chunks are available in the peers determines the order used by the peers to obtain them, this kind of networks is also denominated *data-driven overlays*. Thanks to the high degree of adaptation to this "random-routing" algorithm, data-driven overlays are quite reliable (for example, selfish peers are automatically isolated by their neighbors). The main drawback of the most data-driven overlays is their high latency, which usually is dozens of seconds [37].

PPSP (Peer to Peer Streaming Protocol) is a standard proposed by the IETF [30] for providing VoD. Their authors claim that it is also efficient for the task of broadcasting live media content. Its most comprehensive implementation, the Swirl library [13] (formerly called Libswift), implements a congestion control mechanism called LEDBAT (Low Extra Delay Background Transport) specially designed for real-time applications and has been designed to be used as a new transport protocol (such as TCP or UDP). Like CoolStreaming, Swirl is a pull-based data-driven random-mesh overlay. However, the Swirl latency is much smaller than the CoolStreaming one. Authors report that the startup time is about a few seconds (on average) for an overlay of 128 nodes [59]. However, these results have been obtained in a VoD context, where the file can be sent as quickly as the network allows it. In live media streaming, the average transmission bit-rate used by the peers is smaller than in VoD because the media source performs an inevitable control flow.

The P2PSP modular design is organized in *sets of rules*. Each module, which is introduced below, is specialized in implementing different functionalities.

2.2 LBS (Load Balancing Set)

P2PSP supposes that there is a collection of channels that are broadcasted in parallel. For example, channels in P2PSP are similar to channels in DVB (Digital Video Broadcasting): streams of media that are transmitted in parallel with the rest of the channels.

[1]A buffer map shows which chunks of data has the peer providing it.

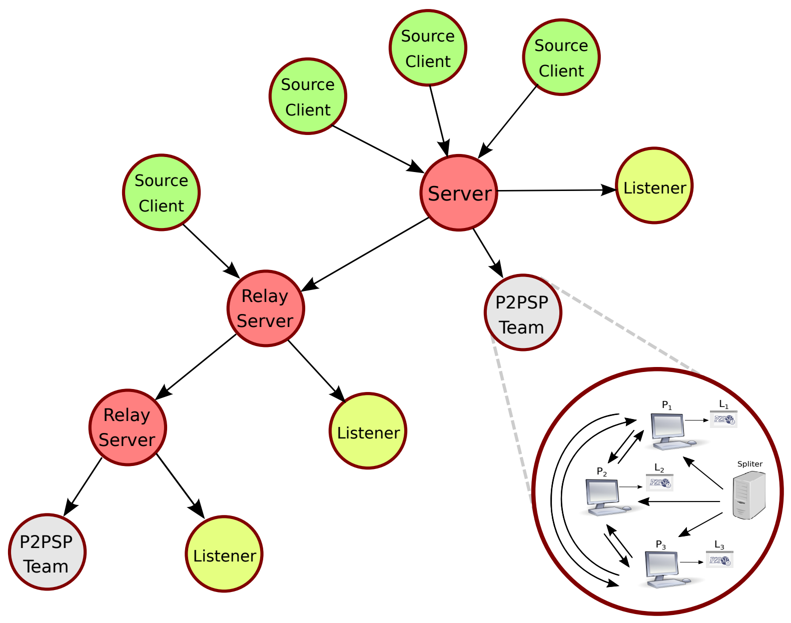


Figure 2.1: A possible data-flow in an hybrid CDN/P2PSP network. Icecast's nomenclature has been used.

The channels are available at one or more streaming *servers*, and each channel has a different URL (Universal Resource Locator), usually expressed as a Web address with the structure:

```
http://server/mount_point
```

P2PSP does not perform data-flow control over the stream. The transmission bit-rate between P2PSP entities is controlled by an external server which provides the stream. Icecast is an example of such a server. Figure 2.1 shows an example of a streaming overlay where several relay servers retransmit a set of channels produced by a set of source-clients, directly or through other servers. As can be seen, a listener (which usually plays de stream) can be replaced by a *splitter*, a P2PSP entity that sends the received stream (a single channel) to a set of P2PSP *peers*, called *team*. Notice that, considering that at least one player can be attached to each peer, the scalability of this hybrid alternative is much higher than the pure C/S architecture.

In a pure CDN system, users request the channels directly to the servers. Unfortunately, this simple procedure has a drawback: normally, users do not know the load or the distance to the servers. This problem can be solved by using a *load balancer*. The listeners, which know the URL of the required channel, connect first to a load balancer which redirects them (with an HTTP 302 code) to a suitable server.

This idea can be extended to minimize the response time of hybrid CDN/P2P structures. When a P2PSP user (who knows an URL of the channel) runs a local peer, it provides the peer with the URL of the channel: a server and a mount point. Then, the peer (as any other listener does) contacts a load balancer which responds with a list of splitters which are broadcasting the channel. In the HTTP request sent to the load balancer, clients must include an `User-Agent:` header line which indicates the type of client. Thus, when the load balancer receives the request from a P2PSP peer, it returns a list of splitters instead of a simple HTTP redirection message. Then, the peer tries to connect with all the splitters in parallel, and the first established connection determines the selected splitter (the rest of connections are closed). If only those splitters with space in their teams answer to the peer, this procedure should select the current "optimal" splitter for the peer in terms of response time.

For the case of the incorporation of new splitters to the network, the procedure is similar. A new splitter (which is instantiated knowing an URL of a channel) contacts the load balancer which returns a list of servers and peers, which are serving the channel. Then, the splitter tries to connect with all of them in parallel and the first successful connection is finally selected. In this case, the load balancer should communicate to the splitter if each entry of the list is a server or a peer, in order to know the type of source.

When a player (listener) connects to a load balancer, if there is a local peer running in the same host or the same private network that the player, the balancer will redirect the player to the local peer.

Finally, all the splitters associated to the same stream should generate exactly the same chunks (content and header). See Section 2.3.

2.3 DBS (Data Broadcasting Set)

This set of rules have been designed to be efficient in transmitting a data-stream from a splitter node to peers in the network when unicast transmissions are used between the nodes of the team. DBS rules are the following:

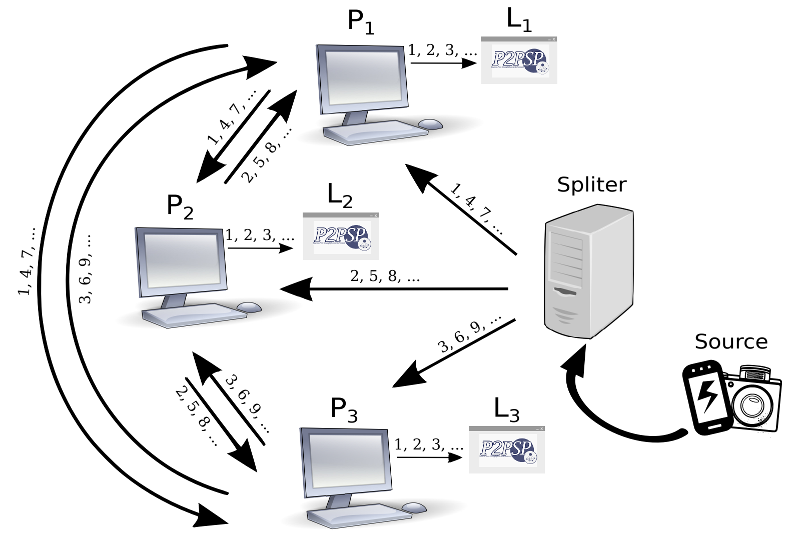


Figure 2.2: A P2PSP team.

1. **Chunk scheduling:** Chunks are transmitted from the splitter to peers, then among peers (see Figure 2.2). The splitter sends the n-th chunk to the peer P_i if

$$(i + n) \bmod |T| = 0, \tag{2.1}$$

being $|T|$ the number of peers in the team. Next, P_i must forward this chunk to the rest of peers of the team. Chunks received from other peers are not relayed.

2. **Congestion avoidance in peers:** Each peer sends chunks using a constant bit-rate to minimize the congestion of its uploading link. Notice that the rate of chunks that reach a peer is a good metric to perform this control in networks with a reasonable low packet loss ratio.

3. **Burst mode in peers:** The congestion avoidance mode is immediately abandoned if a new chunk has been received from the splitter before the previous chunk has been retransmitted to the rest of peers of the team. In the burst mode, the peer sends the previously received chunk from the splitter to the rest of peers of the team as soon as possible. In other words, the peer sends the previous

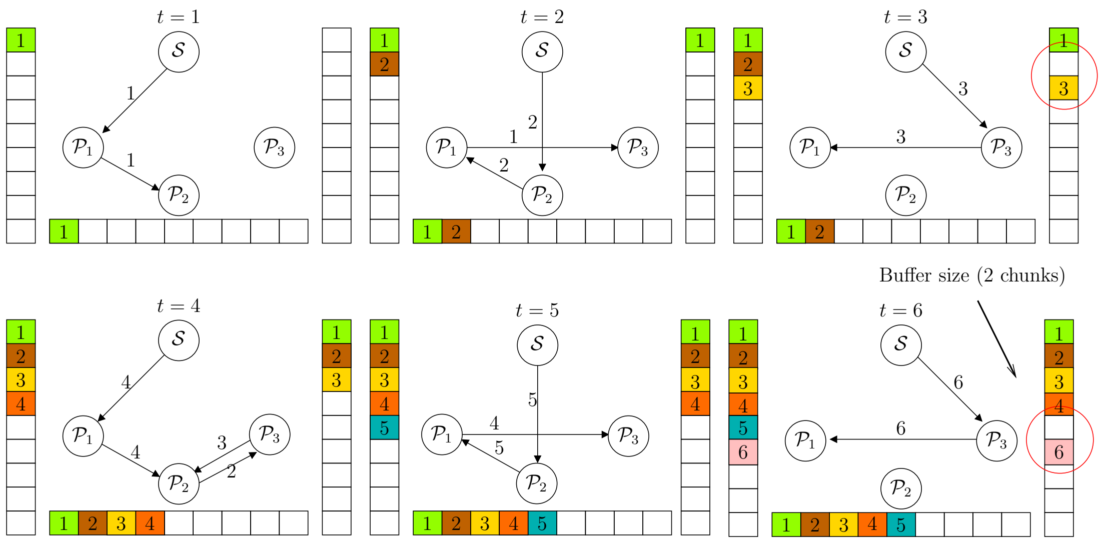


Figure 2.3: Congestion avoidance mode in DBS. Arrows show sent chunks. Buffers show the chunks received by peers, and their size depends on the team size (see red circles).

chunk to the rest of peers of the list (of peers) as faster as it can. Notice that although this behaviour is potentially a source of congestion, it is expectable a small number of chunks will be sent in the burst mode under a reasonable low packet loss ratio.

4. **The list of peers:** Every node of the team (splitter and peers) knows the endpoint $P = (P.\text{IP address}, P.\text{port})$ of the rest of peers in the team. A list is built with this information which is used by the splitter to send the chunks to the peers, and it is used by the peers to forward the received chunks to the other peers.

5. **Peer arrivals:** An incoming peer P_i must contact with the splitter in order to join the team. After that, the splitter sends to P_i the list of peers and the current[1] stream header over the TCP. More precisely, the splitter does:

   (a) Send (over TCP) to P_i the number of peers in the list of peers.

   (b) For each peer P_j in the list of peers:

[1]The stream can be a concatenation of different pieces of audio/video with different headers.

        i. Send (TCP) to P_i the P_j endpoint.

    (c) Append P_i to the list of peers.

The incoming peer P_i performs:

(a) Receive (TCP) from the splitter the number of peers in the list of peers.

(b) For each peer P_j in the list of peers:

    i. Receive (TCP) end endpoint P_j.

    ii. Send (UDP) to P_j a [hello] message.

Because the [hello] messages can be lost, some peer of the team could not know P_i in its introduction. However, because peers also learn about their neighbors when a chunk is received, the impact of these losses should be small.

6. **Free-riding control in peers:** The main idea behind the DBS is that in a large enough interval of time, any peer must relay the same amount of data that it receives. If a (unsupportive) peer can not accomplish this rule, it must leave the team and join another team that requires less bandwidth. In order to achieve this, each peer P_i assigns a counter to each other peer P_j of the team. When a chunk is sent to P_j, its counter $|P_j|$ is incremented and when a chunk is received from it, $|P_j|$ is decremented. If $|P_j|$ reaches a given threshold, P_j is deleted from the list of peers of P_i and it will not be served any more by P_i.

   Notice that this rule will remove from the peer's lists those peers that perform an impolite churn (those peers that leave the team without sending the [goodbye] message).

7. **Monitor peers:** They usually run close (in hops) to the splitter, and play different roles. Among others:

   (a) As a consequence of the impolite churn and unsupportive peers, it is unrealistic to think that a single video source can feed a large number of peers, and simultaneously to expect that the users will experience a high QoS. For this reason, the team administrator should monitor the streaming session. If the media is correctly played by the monitor peer, then there is a high degree of probability that the peers of the team are correctly playing the media too.

(b) At least one monitor peer is created before any other peer in the team and for this reason, the transmission rate of the first monitor peer is 0. However, the transmission rate of the second (first standard) peer, and the monitor peer is:

$$B/2,$$

where B is the average encoding rate of the stream. When the size of the team is $|T|$, the transmission rate for each peer in the team is:

$$B\frac{|T|}{|T|+1}. \tag{2.2}$$

Therefore, only the first (monitor) peer is included in the team without a initial transmission requirement. Notice also that

$$\lim_{|T|\to\infty} B\frac{|T|}{|T|+1} \to B, \tag{2.3}$$

which means that when the team is large enough, all peers of the team will transmit the same amount of data that they receive.

(c) In order to minimize the number of loss reports (see Rule 10 below) in the team, the monitor peers are the only entities allowed to complain to the splitter about lost chunks.

8. **Peer departures:** Peers are required to send a [goodbye] message to the splitter and the rest of peers of the team when they leave the team, in order to stop sending chunks to them as soon as possible. However, if a peer P_i leaves without notification no more chunks will be received from it. This should trigger the following sequence of events:

(a) The free-riding control mechanism (see Rule 6) will remove P_i from the list of peers in peers remaining in the team $\{P_j, i \neq j\}$.

(b) All monitor peers will complain to the splitter about chunks that the splitter has sent to P_i.

   (c) After receiving a sufficient number of complaints, the splitter will delete P_i from his list.

9. **Relation between the buffer size B and the team size $|T|$:** Peers need to buffer some chunks before the playback. However, the main reason of buffering in the DBS is not the network jitter but the overlay jitter. As it has been defined in Rule 1, peers retransmit the chunks received from the splitter to the rest of the team. Also, it has been specified in Rule 2 that peers send these messages using the chunk-rate of the stream. Therefore, depending on the position of a peer P_x in the list of peers of the peer P_y, it can last more or less chunk times for P_y sending the chunk to P_x. We define the chunk time as the time taken for sending a chunk from a peer to another one.

   In order to handle this unpredictable retransmission delay, the peer's buffers should store at least $|T|$ chunks. This means that the team size is limited by the buffer size, i.e., in the DBS module it must be hold that

$$|T| \leq B. \tag{2.4}$$

10. **Chunk tracking at the splitter:** In order to identify unsupportive peers (free-riding), the splitter remembers the numbers of the sent chunks to each peer among the last B chunks. Only the monitor peers will complain about a lost chunk x to the splitter using [lost chunk number x] complaint report messages. In the DBS module, a chunk is classified as lost when it is time to send it to the player and the chunk is missing.

11. **Free-riding control in the splitter:** In this module, it is compulsory that peers contribute to the team the same amount of data they receive from the team (always in the conditions imposed by the Equation 2.3). In order to guarantee this, the splitter counts the number of complaints (sent by the monitor(s) peer(s)) that a peer produces, for all the peers of the team. If this number exceeds a given threshold, then the unsupportive peer will be rejected from the team, first removed from the list of the splitter and next from the lists of all peers of the team (see Rule 6).

2.4 IMS (IP Multicast Set)

IP multicast is available by default in LANs (Local Are Network)s and VLANs (Virtual LANs) [65], but not on the Internet [12]. IMS runs on the top of DBS and provides efficient native IP multicast, where available.

The idea of IMS is simple: all peers in the same LAN or VLAN have the same network address. When a joining peer P_i receives the list of peers from its splitter, first checks if there are neighbors in the same subnet (all of them share the same private network address). For all those peers, P_i uses the IP address `224.0.0.1` (all systems on this subnet), (default) port `1234`, to multicast (only) the chunks received from the splitter. When implementing IMS, all peers in the same local network communicate using this IPM group address and port. The rest of the external peers will be referenced using their public end-points.

2.5 MRS (Massively-lost chunk Recovery Set)

MRS extends DBS to retransmit massively-lost chunks. A massively-lost chunk occurs when a chunk is lost in its way from the splitter to a peer which means the entire team will lose that chunk. MRS should be implemented if error-prone communications are expected, especially if these channels are used by the splitter. MRS is based on the use of monitors (see Sec: 2.3). The idea is: the splitter will resend lost chunks to one or more monitors when all monitors report their lost. To increase the probability of receiving on time the resent chunk (by normal peers), monitors divide by two the number of chunks in their buffers in relation to common peers. Notice that MRS only modifies the behavior of the splitters and the monitors (normal peers does no need to implement MRS).

2.6 ACS (Adaptive Capacity Set)

ACS relaxes the peer's upload requirements imposed by DBS. It should be used in if it is known that some peers can provide the capacity that others cannot, or when we want to mix the CS and P2P models, sending more chunks from the splitter to one or more monitors controlled by the contents provider.

ACS is based on the idea of using the information that the splitter knows about the number of chunks that each peer has lost, to send to more reliable peers a higher number of chunks than to the others peers. In other words, ACS adapts the round-time of each peer to its capacity.

Notice that ACS only affects the behavior of the splitter.

2.7 NTS (NAT Traversal Set)

Most of the peers run inside of “private” networks, i.e. behind NAT devices. NTS is a DBS extension which provides peer connectivity for some NAT configurations where DBS can not establish a direct peer communication. Normally, this problem does not prevent peers from receiving the whole stream. However, if there is a high quantity of peers behind NATs, some peers could lose an enough number of chunks resulting in a low quality of service, or even impossibility the view of the stream for them.

Chapter 5 is devoted to study and analyse port-prediction techniques to be applied in the P2PSP protocol as the NTS set of rules.

2.8 MCS (Multi-Channel Set)

When using MDC [4], SVC [9], or for emulating the CS model, it can be interesting for peers to belong to more than one team. To implement MCS, peers must replicate the P2PSP modules (DBS at least) for each team (channel). We assume that each channel streams different layers of improvement for the same content.

The use of MDC is trivial: the higher the number received descriptions (channels), even partially, the higher the quality of the playback. However, when transmitting SVC media, peers should prioritize the reception of the most important layers.

When a peer is in more than one team, and the teams broadcast exactly the same stream (the same chunks and headers), it could move between teams in a seamless way (without signal loss).

A pure CS service could be provided if the corresponding splitter announces one empty team and sends each chunk so many times as teams (with one peer/team).

2.9 CIS (Content Integrity Set)

The main goal of this set of rules is to face pollution attacks, and one of the most important things is to expel MPs. Some approaches such as the use of TPs, digital signatures, Shamir's Secret Sharing, or moving target defense has been studied to propose a strong content integrity set of rules for the P2PSP protocol.

All the details of content integrity proposals are deeply analyzed in Chapters 3 and 4.

2.10 Conclusions

One important issue for a peer in a P2P overlay is how to find and manage its set of directly connected peers. In this chapter, we have proposed a way to disseminate the stream focusing on minimizing the latency and the protocol overhead. We introduced a modular design which allows taking advantage of the environment where it is running. For instance, using IP Multicast in a LAN, or emulating its behavior on the Internet avoiding congestion and facing free riding. Moreover, other sets of rules specifically designed to secure and to make the protocol more efficient have been introduced. In the next chapters, they are analyzed in depth.

Chapter

3

Security in the P2PSP I: Traditional Networks

Pollution attacks are a challenging security-related problem in peer-to-peer streaming platforms. In this chapter, different variations of these attacks and its combinations are faced. In order to mitigate such attacks, a set of strategies have been proposed in the context of the P2PSP live streaming system. The first prevention strategy is based on the existence of anonymous TPs that detect and report attackers, which are finally expelled. The second strategy increases the security level in the overlay at the cost of a higher computation and communication overhead due to the use of digital signatures. And the last one, enforce peers to collaborate based on the Shamir's Secret Sharing scheme.

3.1 Pollution Attacks

Content integrity is one of the major concerns in most P2P systems when trying to guarantee the quality of service in live-stream transmissions. Pollution attacks [19], also known as stream spoiling, basically consist of a peer or a set of peers modifying the content of the stream. They can be done in different ways. Some of them, and other related attacks are described as follow:

***Persistent attack* [19]:** This attack consists of doing the highest possible amount of damage in the shortest period of time. Therefore, an attacker poisons all the chunks received from the splitter and sends them to the entire team.

***On-off attack* [29]:** In order to avoid to be quickly expelled from the team when a *persistent attack* is being carried out, the attacker can decide to poison only some chunks (for instance, a 10% of the total sent to the team). By acting this way, the attacker is resilient to detection systems using trust management methods, since the attacker would be assigned a high enough level of trust by the team.

***Selective attack*:** This attack consists of poisoning chunks intended for only one peer or a small subset of peers. As in the previous attack, the reason behind this behavior is to be unnoticed by most of the peers and thus to avoid being expelled if a voting system is used in the overlay.

***Collaborative attack* [33]:** Sometimes a single attacker is not able to produce a big damage. However, several attackers may collaborate to produce *selective* and *on-off* attacks to a large set of peers. By doing so, the amount of information obtained by a pollution detection system about an individual attacker is smaller than in single attacker variants. This is the most difficult attack to deal with.

***Hand-wash attack*: [39]** Those attackers that have been discovered or think that they could have been discovered, leave the team and return to continue the attack with another alias.

***Bad-mouth attack* [33]:** It can be used when peers can complain about others in an attacker detection system. Attackers do bad-mouthing by intentionally blame others regular peers for sending poisoned chunks or not sending chunks, with the intention of expelling them.

3.1.1 The impact of pollution attacks

When an MP (Malicious Peer) attacks an HP (Honest Peer) or a TP, both know that they have been attacked. For the sake of simplicity, the worse scenario is supposed: all MPs collaborate and attack anytime they want to, and the splitter does not use a reputation system. Some notation is defined in Table 3.1.

3.1.1.1 Non-selective Attacks

In a non-selective attack, MPs poison[1] chunks relayed to the non-MPs. Non-selective attacks can be classified into persistent and on-off attacks. In the first case, an MP

[1]Or refuse-to-send, which is equivalent for our analysis.

Symbol	Meaning
p	Probability of a TP detects an MP in a round.
A_{C}	Total number of polluted chunks per round independently of their cNumber.
C_M	Number polluted chunks with different cNumber per round and non-MPs.
M	Number of MPs per round.
T	Number of TPs per round.
W	Number of HPs per round.
N	Number of peers in the team (T+H+M).

Table 3.1: Notation used for measuring the impact of attacks.

poisons all the chunks received from the splitter. In the second case, MPs poison a chunk with a probability q. Assuming a memory-less system, in a persistent attack, $q = 1$, $A_{\text{C}} = N - M$ and $C_M = M$. For on-off attack, $p = q$, $A_{\text{C}} = q(N - M)M$ (each MP performs $q(N - M)$ attacks) and $C_M = qM$.

Non-selective attacks are not appealing for MPs in the presence of TPs because there exists a trade-off between the probability of being detected (qT) and the number of attacks per round (A_{C}).

3.1.1.2 **Selective attacks**

$M = 1$ is not interesting to overthrowing the network because it can pollute a maximum of $N - 1$ peers and an attacked peer will have just one polluted chunk out of N, per round. Using selective attack, trying to escape from TPs, will reduce the impact of the non-selective attack.

The probability to be discovered by a TP increases with the number of attacked peers. Therefore, a collaborative-selective attack is more interesting for attackers. Several collaborative-selective attacks can be performed:

***Each one-to-one*:** Each MP attacks one different peer. Its impact is low because $C_M = 1$ and $A_{\text{C}} = M$. The only motivation for this type of attack is to scan the team looking for TPs. The expelled MP can determine who could be the TP(s).

***Each one-to-many*:** Each MP attacks a set of n peers. The motivation is to increase the impact of each one-to-one attack. Different objectives arise:

Increase C_M: MPs attack the same set of non-MPs. So, $A_{\text{C}} = Mn$, $C_M = M$ for n peers and $C_M = 0$ for the rest. The aim of this attack is to avoid playing the stream in the set of attacked peers because degrading the streaming quality will cause that an HP leaves the team.

Increase the scouting of TPs: MPs attack a different set of n non-MPs, minimizing the overlap among them. The altered chunks/round $A_{\text{C}} = Mn$. When the sets do not overlap, $C_M = 1$ for Mn peers. The aim is to narrow the TPs location when $M << N$. Otherwise, *each one-to-one* attack is preferable.

***Many-to-one*:** All MPs attack just one peer. In this case, $A_{\text{C}} = M$ and $C_M = M$ for the attacked peer and $C_M = 0$ for the rest. Its aim is to increase the probability that the attacked peer leaves the team due to the low quality of the stream. MPs may reduce their exposure if a new target is not selected while the current one is still on the team.

***Many-to-many*:** Each MP performs a *one-to-many* attack, but MPs collaborate to determine the targets.

MPs have a difficult multi-objective optimization problem: i) maximize the number of detected TPs, ii) minimize the number of MPs (cost), iii) minimize the number of expelled MPs, iv) maximize C_M in HPs, and v) perform previous tasks in a minimum number of rounds.

TPs have also a multi-objective optimization problem: to i) minimize the number of MPs (this minimize C_M in HPs), ii) minimize the number of TPs (cost) and iii) performs previous tasks in a minimum number of rounds.

The difficulty of both problems increases with the existence of churn. Several solutions can be devised, and each one will present a possible countermeasure. Following, we show the strategies studied here as possible non-optimal solutions.

3.2 Related Work on P2P Security

Vulnerability and MP behavior are some of the major concerns in P2P overlays. For this reason, these topics have been studied extensively. In the specific scenario of live streaming, the dominant approaches to fight against attacks are: peer polling, non-repudiation systems, trust management systems, and incentive mechanisms.

Following, we analyze the advantages and drawbacks of the most frequently used solutions for free-riding and pollution attacks.

3.2.1 Peer polling

Peer polling is the most simple idea for fighting against MPs. The votes of the majority about a given peer to determine its goodness is a widely accepted solution in many systems [6].

- Advantage: It is carried out easily.
- Drawback: In case the attacked peers are less of the half of the team, it is vulnerable to a selective attack. For instance, several MPs may attack to few HPs and behave correctly with others. Then, it is difficult to determine who is actually performing the attack.

3.2.2 Non-repudiation systems

In a non-repudiation system, a TTP (Trusted Third Party) is used to ensure that a communication was correct. There exist non-repudiation methods without TTPs [24] but they consider that both parties are interested in the transmitted content. Moreover, the number of necessary messages to provide non-repudiation is usually high.

- Advantage: It is a deterministic method and it can prove who says the truth.
- Drawbacks:
  1. The use of TTPs is not in consonance with the distributed computing philosophy of P2P.
  2. MPs must be interested in receiving the content. So, these systems are useless for combating free-riding.

3.2.3 Trust management systems

They consist in gathering information about peers and compute their trust value taking into account their previous behavior [29, 44, 61, 73]. In this way, a peer could determine who the best one is to share data with.

- Advantage: All peers are aware of the behavior of others.
- Drawback: Like polling systems (see Section 3.2.1), it can produce false positives and false negatives (i.e., spelling HPs and not expelling MPs) under selective attacks.

3.2.4 Incentive mechanisms

This is one of the best solutions for most of the P2P overlay networks. It is used in well known P2P systems, such as BitTorrent [10]. Peers reciprocate uploading to peers which upload to them, following a tic-for-tac approach [11].

- Advantage: It performs an efficient usage of all the bandwidth of each peer, ensuring that they maintain fruitful connections.
- Drawback: It is not applicable to a push-based fully connected mesh scheme because MPs does not need the chunks from others in order to pollute the chunks they relays.

Most of the defenses are based on trust management by gathering the reputation information from subsets of neighbors in the network. A framework with a set of TPs to monitor the bandwidth usage of untrusted peers in the system to prevent free-riding was studied in [14]. A detection algorithm for a set of TPs was considered in [23], where peers inform to the assigned TPs about the correct reception of a chunk. Then, the set of TPs can identify MPs by solving an inference problem. In P2PSP, TPs have their own information about the team because it is a fully connected overlay network. An attacker is expelled by the splitter based on the information received from TPs. The splitter expels a peer by removing it from its list of peers and therefore it will receive chunks anymore.

Almost all studies in the literature about defense mechanisms fighting against free-riding and pollution attacks in P2P systems are applied to pull-based overlays, probably because they are more widespread due to their better performance in networks with a huge number of nodes. However, we focus in a push-based overlay as a “last mile” solution [41] because it is more efficient in terms of bandwidth and delay.

3.3 Detecting attackers using TPs

This section presents two different strategies aiming to mitigate the impact of pollution and related attacks by combining trust management, hashing/signatures and TPs. Each strategy defines a set of rules, specially designed for the P2PSP system. The first strategy, denoted by STrPe (Strategy based on Trusted Peers), is a simple approach with a low data overhead in the overall operation of the team. The only difference with respect to an pollution-unaware P2PSP system is an extension of the splitter functionality and the inclusion of anonymous TPs who transparently monitor the behavior of regular peers in the team. Regular peers see a TP as another regular peer.

The second strategy, denoted by STrPe-DS (Strategy based on Trusted Peers and Digital Signatures) is an extension of the first one, where a digital signature of a chunk allows to detect attackers. STrPe-DS generates more data overhead than STrPe but the performance of the defense is greatly improved.

Those attacks detected by STrPe are also detected by STrPe-DS, but not the other way around. Therefore, the most appropiate strategy should be used depending on the risk to be assumed. This decision is usually made before the deployment but it is also possible to change it once the P2PSP overlay is running.

3.3.1 Strategy based on Trusted Peers (STrPe)

STrPe has been designed to maintain the simplicity characterizing the P2PSP system. By including TPs into the team, an attacker sending a poisoned chunk to some TP is detected and expelled from the team. For this reason it is important to maintain the anonymity of TPs among regular peers. The behavior rules are:

1. Only the splitter knows who the TPs in the team are. It is possible that all peers are TPs except the attacker.

2. Each TP creates a hash for each received chunk, including the chunk number and the endpoint of the source of the chunk. Depending on the computational power available in the TPs, all chunks or a random subset of them may be processed in each round.

3. TPs send the hashes to the splitter, who checks whether the chunks have been altered by comparing them with those hashes calculated when the chunks were delivered to the team. The splitter only listens to TPs for this task.

4. The splitter knows the peer in charge of relaying a given chunk and it knows who altered a chunk when an invalid hash has been received from a TP. Any exposed attacker is expelled from the team by removing it form the list of peers in the splitter (this implies that no more chunks will be send to it). In order to ensure that the attacker is removed from all lists of peers as soon as possible, the splitter sends a expulsion message containing the endpoint of the attacker to all peers in the team.

3.3.2 Strategy based on Trusted Peers and Digital Signatures (STrPe-DS)

The STrPe-DS has been designed to mitigate the *selective attack* and to identify poisoned chunks by using digital signatures. So, any peer is able to know if a chunk was poisoned. A peer also knows if a chunk is relayed by a different peer than the one in charge of it. Some interesting notation used from now on is defined in Table 3.2. Rules defining STrPe-DS are:

1. A peer receives the public key of the splitter, plus the other necessary information, when a peer joins the team.

2. For each chunk, the splitter sends a message with the chunk, its number (*nChunk*), the destination address (*dst*) and a digital signature

   $\{chunk, nChunk, dst, \mathrm{S}_{priv}(H(chunk + nChunk + dst))\}$,

   where H is a hash function and S_{priv} is the private key of the splitter.

3. When a peer receives a message, it performs the following steps:

Table 3.2: Notation used in P2PSP with TPs.

Symbol	Meaning
n	Current number of peers in the team.
L	List of peers.
$\|L\|$	Number of elements in L.
C	List of chunks.
S	Splitter.
S_{pr}	Private key of the splitter.
S_{pu}	Public key of the splitter.
H	Hash function.
E_K	Encryption using the secret K.
D_K	Decryption using the secret K.
$\|\|$	Concatenation.

(a) Check whether *dst* matches the address of the sender. Notice that this action is vulnerable to the well known *spoofing attack* [5]. The next step is performed only if this one is successful.

(b) Check the correctness of the hash value in the message.

If any of previous checks fails, the sender is removed from the list of peers of the current peer.

4. The splitter periodically requests (and the peers serve) the list of removed peers (since the previous request). In this way, a *DoS (Deny of Service) attack* by sending removed peers to the splitter at high rate is avoided. TPs are the only ones that can send the list of removed peers to the splitter as soon as they are detected.

5. Peers removed by any TP are directly expelled by the splitter (see point 4 in Section 3.3.1).

6. The splitter can decide to expel a peer based on the information received from honest or MPs. So, a typical approach here is to establish a trust value for each peer depending on several aspects, such as the number of complaints received

about a peer during a given period of time, the number of affected peers by a possible attack, the age of the peers in the team, etc.

Any exchange of information between the splitter and peers should be authenticated by a digital signature in order to avoid *spoofing* attacks.

Following, algorithms for the splitter and a peer in the P2PSP system with TPs are shown.

Algorithm 3.1 TP:S.

```
Require: $L$, $S_{pr}$, $S_{pu}$
1: $i \leftarrow 0$; $j \leftarrow 0$
2: while $S$ is alive do
3:     $C_j \leftarrow$ getNextChunk()                 #get next chunk from the source
4:     $m = \{C_j, j, P_i, S_{pr}(H(C_j||j||P_i))\}$
5:     sendMessage($m$, $P_i$)                      #send message m to peer $P_i \in L$
6:     $i \leftarrow (i + 1)$ mod $n$
7:     $j \leftarrow j + 1$
8:     if TP complains about $P_x$ then
9:         expelMP($P_x$)                           #at random time
10:        recoverLostChunks()
```

- Algorithm TP:S (using Trusted Peers - Splitter's algorithm):
  - The splitter S sends a different message (m) to each peer (see Alg. 3.1, line 5). The message includes the chunk C_j, the chunk number j, the destination address P_i and a digital signature:

$$m = \{C_j, j, P_i, S_{pr}(H(C_j||j||P_i))\} \tag{3.1}$$

  - Peers reported by any TP in a secure way will be expelled by S after a random time, in order to disassociate the expulsion with the action taken by the attacker (see Alg. 3.1, line 9).

TPs report lost chunks to S as soon as possible. If all TPs complain about a given chunk, the splitter re-sends this chunk to a *monitor peer* (a reliable peer that is controlled by the team administrator) to relay it.

Algorithm 3.2 TP:P_i.

```
Require: $S$
1: ($L$, $S_{pu}$) ← receiveFrom($S$)
2: mine ← $\emptyset$
3: while player is alive do
4:    $(m,$ sender$)$ ← listenMessage()                #wait for a message
5:    ($C_j$, $j$, $P_i$, sign) ← unpack($m$)              #unpack message m
6:    if sender == $S$ then                            #$S$ send a message to relay
7:       sendBurstMode(mine)    #send the previous message to be relayed to pending peers in
         burst mode
8:       mine ← $m$                                 #my new message I have to relay
9:    else
10:      if sender == $P_x$ and $H(C_j||j||P_x)$ == $S_{pu}$(sign) then
11:         sendToPlayer($C_j$)
12:      else
13:         removePeer($P_x$)                          #$L = L - \{P_x\}$
14:         if I am a TP then
15:            sendComplaint($S$, $P_x$)               #complain about the sender
16:   $q \leftarrow (q+1) \bmod |L|$
17:   sendMessage(mine, $P_q$)                    #relay my message to $P_q \in L$
```

- Algorithm TP:P_i (using Trusted Peers - Peer's algorithm):
  - A peer receives S_{pu} from S, plus other necessary information, when a peer joins the team.
  - When a peer receives a message from another peer, it performs the following steps:
    1. Check whether P_i matches the address of the sender. Notice that this action is vulnerable to the well-known *spoofing attack* [5].
    2. If 1) was satisfied, it checks the correctness of the hash value in the message.

  The sender is removed from L of the current peer if 1) or 2) fails.

– The relay of a message is triggered by the reception of another message from another peer (*congestion-avoiding mode*), in order to avoid buffer overflow (see Line 17)). However, upon a message reception from S, the message is relayed in *burst mode* to pending peers (see Line 7).

3.3.3 Discovering and hiding TPs

MPs must devise an algorithm to detect TPs. In a simple approach, each MP increases the set of attacked peers by just one per round. MPs collaborate to attack a peer that has not been attacked before. Under the assumption that a TP informs to the splitter as soon as it is attacked, the MPs team can collaborate to make out who is a TP when an MP is expelled.

For $T = 1$, the TP can be discovered in $\lceil \frac{N}{M} \rceil$ rounds. The expelled MP will inform to others MPs in the team and MPs will attack the discovered TP anymore. Then, the impact of the attack would be $p = 0$, $A_{\text{C}} = N - T - M$ (all HPs), $C_M = M$ in the worst case.

To overcome the previous algorithm, the splitter will expel an MP in one of the next rounds (at random). Moreover, a TP should leave the team in next rounds (at random) when it informs to the splitter about an MP, and another TP with different identity should be included in the team. In this way, MPs have more difficulties to figure out who a TP is.

3.3.4 Analytical Results

Defense mechanisms in P2P overlays can be evaluated in several ways. The preferable is to test the system in a real scenario, but the cost of these experiments can be unaffordable. Another possibility is to analyze a set of simulations with different combinations of attack/defense schemes. We will address them in next sections. Here we perform a theoretical study of the advantages and drawbacks of STrPe and STrPe-DS strategies.

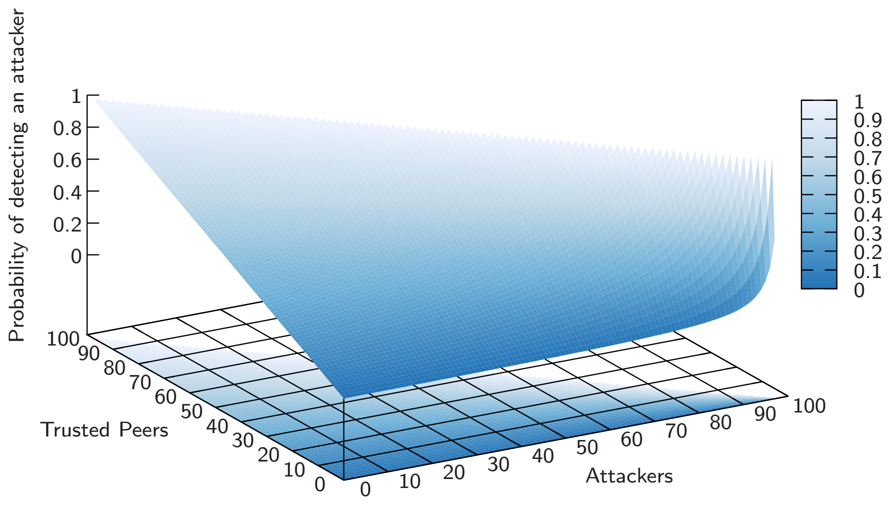


Figure 3.1: Probability to detect an attacker depending on the percentages of trusted peers and attackers in the team.

3.3.4.1 **STrPe**

Regarding the communication overhead, only the communications between the trusted peers and the splitter suffer it in this strategy. TPs need to send to the splitter a message digest for each received chunk from other peers, this implies a total overhead $O = sH \times Cps \times nTP$, where sH is the size of the hash, nTP is the number of trusted peers and Cps are the chunks per second. For instance, let suppose that a video, whose bit-rate is $1{,}024$ Kbps, is being streamed and that the chunk size is $1{,}024$ bytes. So, at least 125 chunks are being transmitted per second. If hash function generates a digest of 224 bit, the overhead per chunk is 224 bits. Thus, the communication overhead between a trusted peer and the splitter is $125 \times 224 = 28{,}000$ bps $\simeq 27.3$ Kbps. In this example, the total overhead in the team due to the defense strategy would be 27.3 Kbps $\times\, nTP$.

Concerning the probability of detecting an attacker, Figure 3.1 shows it as $P = nTP/(N-A)$, where N is the team size and A is the number of attackers. The number of attackers and TPs represents a percentage of the size of the team. In general, the chance to detect an attacker increases with the number of trusted peers in the team.

A *persistent* and *on-off attacks* are quickly detected by the splitter. A *bad-mouth attack* makes little sense in STrPe since only TPs are able to inform about hashes to the splitter. However, the main vulnerability of STrPe is that TPs must remain anonymous. In case of being discovered, the system will not be resilient to *selective attacks* where attackers do not pollute chunks for TPs. Finally, regular peers do not know if a chunk has been polluted because the chunks are not digitally signed.

3.3.4.2 **STrPe-DS**

Whereas only TPs have communication overhead in previous strategy, the whole team suffer from it in this strategy, because all messages contains not only the chunk but also additional information (see Sect. 3.3.2). Using the same example of the previous section, if a RSA signature with $1{,}024$ bit key is used (it seems to be fast enough for our purpose, according standards signature times [16]), the overhead per chunk is $1{,}072$ bits ($1{,}024$ bits for signature + 48 bits for dst). Thus, the communication overhead is $125 \times 1{,}072 = 134{,}000$ bps $\simeq 130.8$ Kbps. However, in this strategy, to calculate the total overhead in the team is more difficult because it depends on the size of the removed peers by request made by the splitter, the frequency of such requests, the number of attackers and the number of the trusted peers.

Regarding the detection of attacks, when a *persistent attack* is carried out over the entire team, it is quickly detected by any TP (see Figure 3.2).

A *on-off attack* is meaningless because peers remove the attackers from their lists as soon as they receive an incorrect message, according to the digital signature of the splitter. This attack behaves as a *persistent attack* in a relatively large period of time.

Performing a *bad-mouth attack* by only one attacker is not usually enough to expel a regular peer in trust based systems. Additionally, the attacker will usually have more complains than any attacked peer and could finally be detected. According to the P2PSP system, a peer A removes a peer B from its list of peers because it does not receive chunks from B or the chunks received are incorrect. This results in B also removes A from its list of peers given that B is not receiving chunks from A anymore. In any case, it is not possible to know who is the attacker.

Expelling a legitimate peer by means of a *bad-mouth attack* requires a combination with *selective* and *collaborative attacks*, and the set of selected peers to be attacked must be small in comparison with the set of attackers. In the absence of TPs, the team can run out of HPs after several repetition of previous combination of attacks.

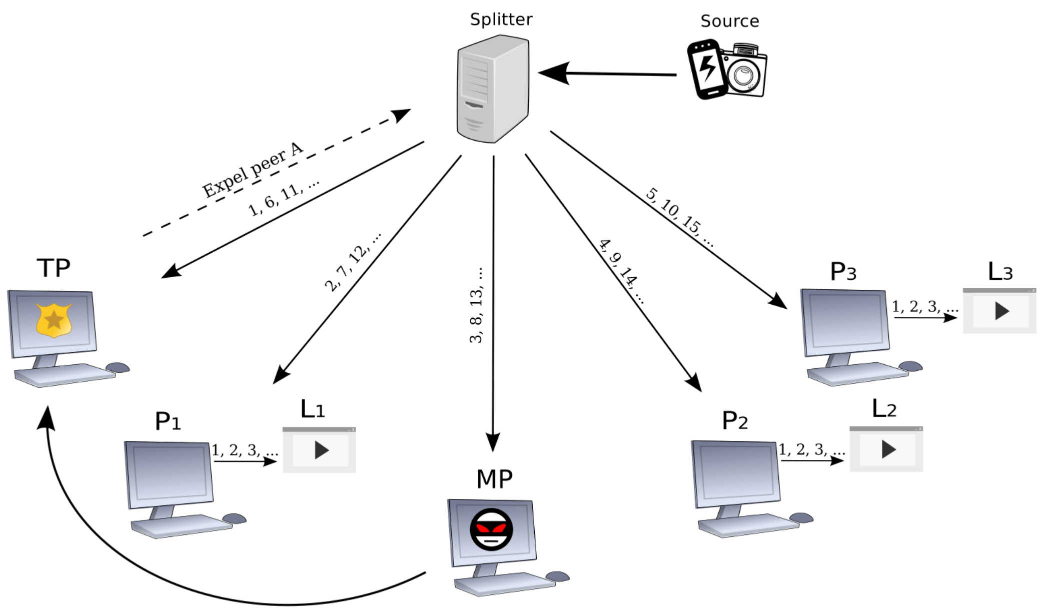


Figure 3.2: A P2PSP using STrPe-DS, where a poisoned chunk is received by the TP, which immediately informs to the splitter.

Therefore, not only the rejection-threshold in the splitter or the trust value of peers but also the number of trusted peers in the team are important.

A *hand-wash attack* is difficult to deal with, if the attacker can guess the value of the rejection-threshold. The rejection-threshold is established by the splitter to minimize the damage of the possible attacks and the possibility to expel a polite peer. TPs should perform a kind of *hand-wash strategy* in order to reduce the probability of being detected.

To the best of our knowledge, there is no effective defense against a combined *selective*, *collaborative*, *bad-mouth* and *hand-wash attacks* in free access P2P live streaming protocols.

3.3.5 Experimentation

The experimentation is performed to measure the number of affected peers during the attack, the number of affected chunks (per peer and round), the average expulsion time for MPs, the average number of HPs and MPs and the state of the buffer of HPs. Time is expressed in rounds.

3.3.5.1 **Test bed**

For all the experiments we used a simulator specifically coded for evaluating different aspects of the P2PSP protocol. Both, the protocol and the simulator are open source and they are available at [71, 72]. They were run on a Unix machine (Ubuntu server) with 2.3TB of RAM and 8 Intel Xeon E7 8860v3 (16 cores) processors.

Churn:

1. The initial team consists of a splitter and an initial number of HPs (#IHPs).
2. To avoid any tracking of TPs, they will be incorporated into the team at random, mixed with the arrival of the rest of peers (MPs and HPs) up to reach a #TP$_{max}$.
3. As #TP$_{max}$, the values of the maximum number of HPs (#HP$_{max}$) and MPs (#MP$_{max}$) limit the number of entries of HPs and MPs in the team along the simulation.
4. A new peer (MP or HP) arrives with probability P_{in}. The peer will be HP with probability P_{HP} and MP with probability P_{MP}. Additionally, when a new TP is needed, it joins the team also with probability P_{in}.
5. Peers (all types) stay in the team a period of time modeled by a Weibull distribution with the shape parameter $a = 1$ [69].
6. Events in peers are triggered when a new chunk comes to them. A peer that spends its Weibull time will leave the team with probability P_{out} in next event.

MPs behavior:

1. MPs always attack (persistent, selective and collaborative attack).
2. An MP should not attack over 50% of the team to reduce the probability of being detected.

3. MPs should scout for TPs. We denote the set of peers Not Attacked Yet as NAY. Only one MP attacks to a NAY peer. NAYs peers that were not attacked by expelled MPs are attacked before others.
4. Additionally, MPs should attack peers as much as possible. Then, every MP also attacks peers that have been attacked more than MPTR (Malicious Peer Test Round) by other MPs.
5. MPs share the list of attacked peers and how many rounds an MP was attacking each of them. Thus, they know the NAY and peers attacked more than MPTR rounds.

HP behavior under attacks:

1. They will leave the team if the Exponential Smoothing of Chunks Loss Rate (ESCLR) is greater than ESCLR_{max}. ESCLR defined as:

   $$\text{ESCLR}_i = \alpha\text{CLR}_i + (1-\alpha)\text{ESCLR}_{i-1} \tag{3.2}$$

   where CLR_i is the chunks loss rate measurement in last round.

   CLR is defined as:

   $$\text{CLR} = \frac{\text{polluted/lost chunks in B}}{|\text{B}|} \tag{3.3}$$

   where B is the buffer of the peer and $|\text{B}|$ is the size of B.

TPs behavior under attacks:

1. TPs inform to the splitter about attacks as soon as they are attacked.
2. TPs, as HPs do, will leave the team if the $\text{ESCRL}>\text{ESCLR}_{max}$.
3. A TP will leave the team after complaining about an MP with probability P_{TPL} in order to avoid its detection by MPs.

Splitter behavior against attacks:

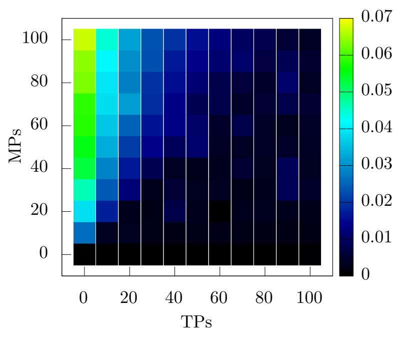
1. A reported MP will be expelled in current round with probability P_{MPL}.

3.3.5.2 **Experimental results**

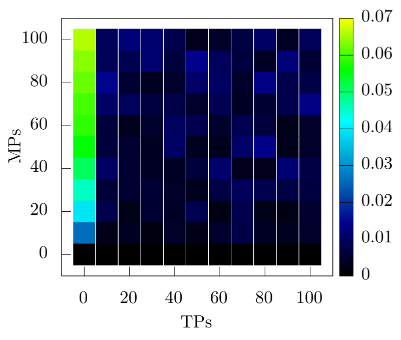
The main aim of this study is to know if it is feasible to mitigate pollution attacks with previous strategies. We focus on the defense side and the tune of P_{MPL} and P_{TPL} values. For all experiments $P_{in} = 1$, $P_{out} = 0.25$, $P_{MP} = P_{HP} = 1$, $MPTR = 2$ and ESCLR$_{max} = 1$ (peers stay on the team even when the stream has a low quality). #TP$_{max}$ and #MP$_{max}$ values were taken from the 100 combinations of values in $\{0,\ 10,\ 20,\ \ldots,\ 90,\ 100\}$. The simulation stops 100 rounds after the requested number of peers (#HP$_{max}$ + #TP$_{max}$ + #MP$_{max}$) got in the team. Additionally, P_{MPL} and P_{TPL} values used in each of the previous combinations are shown in Table 3.3. Therefore, the number of experiments is 1100. Five executions have been carried out for each experiment, and after discarding the outliers, the average of the remaining three values was returned.

Exp	1	2	3	4	5	6	7	8	9	10	11
P_{MPL}	100	100	100	100	100	100	80	60	40	20	0
P_{TPL}	100	80	60	40	20	0	100	100	100	100	100

Table 3.3: Parameter values for the different experiments.



(a) Exp 1.



(b) Exp 6.

Figure 3.3: Numerical results. Each small square is a combination of #TP$_{max}$ and MP$_{max}$ values. Right column shows ESCLR values. The darker the better.

All the instances in Table 3.3, except Exp. 6, show similar results to Exp. 1 instance, shown in Figure 3.3a. Therefore, variation in P_{MPL} and P_{TPL} has not a great impact on the results. In general, when #TPs≥#MPs the ESCLR is low.

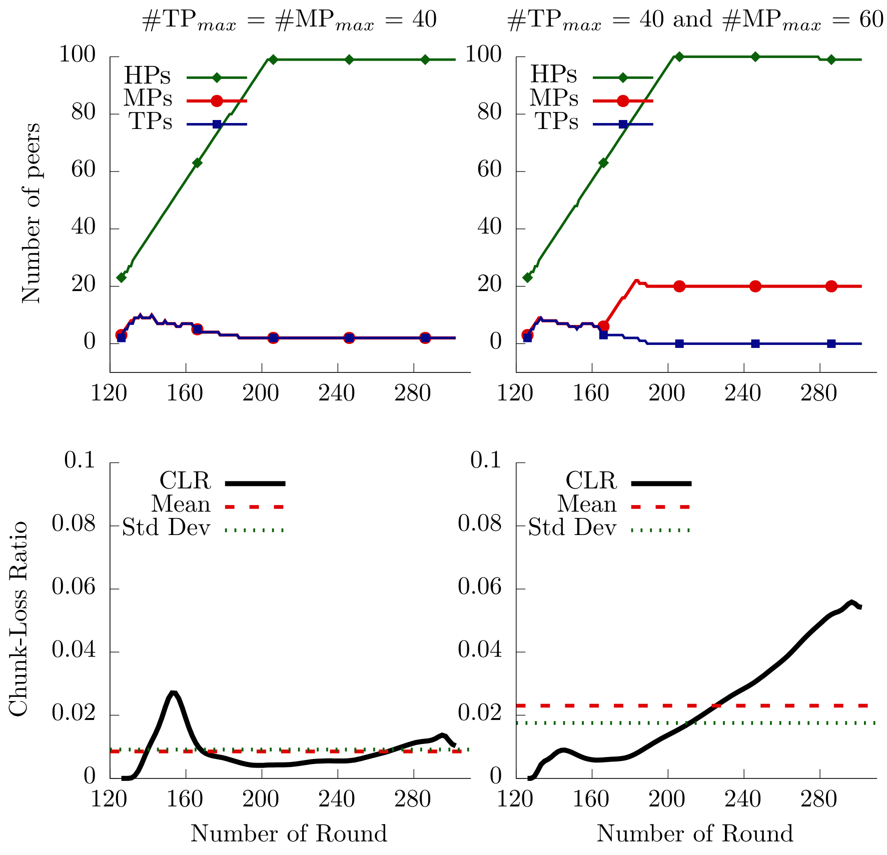


Figure 3.4: Examples of changes in the composition of the team for Exp. 1 (up) and their consequences in the chunk-loss (down).

Figure 3.3b shows the results for Exp. 6 with P_{TPL}=0. Now, TPs do not leave the team during the entire simulation. The value of MPTR=2 and MPs sharing their “safe to attack” list including TPs produce that all MPs attack TPs and consequently they are expelled. A solution for MPs is to increase the number of attacked peers just by one in each round and a different one by each MP. In this way, the number of MPs attacking a TPs due to the “safe to attack” report of other MPs is much smaller than the entire MP set.

Figure 3.4 depicts the state of the team and the chunk-loss ratio for the Exp. 1 and two different combinations of #TP$_{max}$ and #MP$_{max}$. In order to have a stable state in the team before the attack, the initial rounds, where only HPs are allowed, are not plotted). These results show that TPs can expel almost all MPs when their number is similar (all MPs if TPs are a majority), but in general, TPs cannot do it when they are a minority. In such case, the remaining number of MPs is approximately $\#\text{MP}_{max} - \#TP_{max}$. Therefore, other MPs control mechanisms, such as reputation systems, well behavior rewards, etc. should be added to the use of TPs. Notice that most of these systems have been studied for pull based P2P systems, but there exist few studies for push based systems, like the P2PSP [38].

3.3.6 Conclusions

In this section two different strategies to mitigate pollution attacks in the P2PSP system are studied: (1) a strategy with low computation and communication overhead, which is designed for resource constrained devices and (2) an extension of the first strategy that increases the defense level against pollution attacks at the expense of introducing a higher overhead.

Attackers and defenders have to solve a multi-objective optimization problem and each decision made by one part may have a countermeasure by the other part. A possible collaborative-selective algorithm performed by attackers is devised with two main objectives: i) to discover trusted peers and ii) to pollute as much as possible the buffer of HPs without been expelled. At the defense side, the proposed algorithms try to hide the trusted condition of a peer to attackers. Both algorithms are interesting because the following question could arise: Who are the good/bad guys? The answer depends on the scenario. In a normal one, the stream belongs to defenders. But good guys are the attackers when bad guys perform an illegal public free rebroadcast in a P2P network of their private stream, i.e., the stream belongs to attackers.

The experimental results show that using only trusted peers as a defense strategy is not appealing when the number of MPs can be large. Therefore, the improvement of the policies and additional techniques should be addressed.

Assuming a large number of MPs, but keeping most of the team with honest peers, we could enforce to MPs to collaborate in order to avoid being expelled. This approach is taken and analyzed in the next section.

3.4 Enforcing Peers to Collaborate using Shamir's Secret Sharing

As we have seen in previous sections, in P2PSP, as in many other P2P systems, MPs can perform collaborative pollution and free-riding attacks. The use of trusted peers as the only defense strategy requires a similar number of TPs which implies a high quantity of resources in the defense side. Here, we propose a solution based on Shamir's Secret Sharing (SSS) and the use of TPs. Under the assumption that the number of MPs is smaller than the half of the total, our proposal forces any peer to relay unpolluted content to honest peers in order to not to be expelled. We show the overhead added to the protocol, its strengths, its weaknesses, and outline possible solutions for these weaknesses.

3.4.1 T-Shared Method

In this section the following questions are answered:

Question 3.1. *Could we alleviate the content pollution in P2PSP using SSS?*

SSS [66] has been used before in other P2P networks but with a different objective than the current one. Mainly, SSS has been used to keep the content private and secure [26]. Instead, we intend to use it to force the collaboration of peers in a public streaming network.

Question 3.2. *What would be the optimal value of the SSS threshold t?*

The main idea is very simple: *"if you want to remain in the team you must have a good behavior"*, but it is very difficult for S to determine who is having a good behavior with others. Therefore, we relax this requirement by adding to the sentence *"with at least t peers"*.

The SSS function, SSS(t, n), generates a secret k and a set of n shares. One peer having t out of n shares can derive k [66]. The probability of two SSS(t, n) producing the same key should be very low.

In Alg. 3.2, a peer only relays a chunk to peers that have behaved correctly. A HP P_i continues communicating with peer P_q only if:

- it has not lost chunks from P_q, and
- all received messages from P_q always have a valid signature (see Equation (3.1) and Alg. 3.2, line 10).

The idea is to cipher the signature of each message (containing a chunk) from S to P_i with a SSS key k_i. The relayed messages among peers must have a valid signature, i.e. the signature has not to be ciphered. Therefore, the peer relaying the message have to get at least t shares to derive k_i in the previous round. Below, it is shown how shares for each k_i are provided by S to peers. We define a round r as:

- For S: The n iterations needed to sent a message to each peer P_i, $i = 1, \ldots, n$.
- For a peer: The $|L|$ iterations to relay the message received from S. $|L| < n$ is possible on a peer, because MPs may be removed from its list of peers.

In round r, a peer P_q will relay a valid message and also piggyback a share needed by the destination peer P_i to help P_i to derive k_i^{r+1} only if:

- P_q could obtain at least t_{r-1} shares in the $r-1$ round from others peers in order to decipher the signature (derive k_q^r) and
- P_i always sent valid messages to P_q.

By definition,

$$n_r = \#\text{MP}_r + \#\text{HP}_r. \tag{3.4}$$

The number of TPs (#TPs) is included in the number of HPs in round r (#HP$_r$).

We impose the following condition:

$$\#\text{MP}_r < t_r \leq \#\text{HP}_r, \tag{3.5}$$

because in this way:

- Any correctly behaved peer can obtain at least t_r shares from HPs.

- MPs are not able to derive a key by themselves. Even if MPs communicate the received shares from S among them, each MP will still need at least, $t_r - \text{\#MP}_r$ shares. Therefore, each MP is forced to send a valid message to

$$t_r - \text{\#MP}_r \tag{3.6}$$

  HPs. Otherwise, it cannot send a valid message to another peer, TPs will complain about it to S, and it will be expelled from the team. This answers the Question 3.1.

Equation 3.5 answers the Question 3.2. The best t_r value is $t_r = \text{\#HP}_r$, but it is difficult to know this value, as it was shown in Sect. 3.4.1. From Eqs. 3.4 and 3.5 one can derive

$$\text{\#MP}_r < \left\lceil \frac{n_r}{2} \right\rceil . \tag{3.7}$$

So, for systems satisfying Equation (3.7),

$$\text{\#MP}_r < \left\lceil \frac{n_r}{2} \right\rceil \leq t_r \leq \text{\#HP}_r. \tag{3.8}$$

Hereinafter, $t_r = \left\lceil \frac{n_r}{2} \right\rceil$ is used.

Following, it is explained how S generates the shares, sends them to peers and how peers share them.

At the beginning of round r, all expelled MPs in round $r-1$, the peers asking to join in round $r-1$, and HPs peers outgoing after completing their $r-1$ round are known. Therefore, n_r is well determined. Then, S generates a vector $K^{r+1}_{(n_r)}$ of n_r keys and a matrix $A^{r+1}_{(n_r \times n_r)}$ of shares by n_r calls to SSS(t_r, n_r). Each call results in a K^{r+1}_i and a row $A^{r+1}_{i,*}$ with n_r shares to derive K^{r+1}_i.

In round r, S sends to peer P_i a message $eSP^r_{j,i}$

$$eSP^r_{j,i} = \{eCH^r_{j,i},\ SH^{r+1}_i\}. \tag{3.9}$$

$$eCH^r_{j,i} = \{C_j, j, P_i, r, E_{K^r_i}[S_{pr}(H(C_j||j||P_i||r))]\} \tag{3.10}$$

$eCH^r_{j,i}$ has the following content:

- The chunk C_j.
- The index j of the chunk in the stream.

- The peer P_i in charge to relay C_j.
- The round r.
- The signature of previous fields encrypted with the Shamir key for P_i and round r ($E_{K_i^r}$).

The second part of the $eSP_{j,i}^r$ (see Equation (3.9)) is:

$$SH_i^{r+1} = \{\{SH_i^{r+1}\}_q,\ q = 1,\ \ldots,\ n_r\}, \tag{3.11}$$

with

$$\begin{aligned}\{SH_i^{r+1}\}_q =& \{P_i, P_q, r+1, A_{q,i}^{r+1},\\ & S_{pr}(H(P_i||P_q||r+1||A_{q,i}^{r+1}))\}\end{aligned} \tag{3.12}$$

So, SH_i^{r+1} is a vector with n_r elements. Each element $\{SH_i^{r+1}\}_q$ contains:

- The peer P_i who has to send the share $A_{q,i}^{r+1}$ to peer P_q.
- The peer P_q who can use the previous share to derive K_q^{r+1}.
- The round $(r+1)$ the share is created for.
- The share.
- The signature of previous fields with the splitter private key (S_{pr}) to allow P_q to validate the contents.

Once P_i has received $eSP_{j,i}^r$ from S, in order to P_i not to be removed from the list of peers of P_q, it has to send to P_q a valid message

$$PP_{i,q}^r = \left\{CH_{j,i}^r,\ \{SH_i^{r+1}\}_q\right\}. \tag{3.13}$$

Where $CH_{j,i}^r$ is equal to $eCH_{j,i}^r$ but with the splitter's signature not ciphered by K_i^r:

$$CH_{j,i}^r = \{C_j, j, P_i, r, S_{pr}(H(C_j||j||P_i||r))\} \tag{3.14}$$

In this way, P_q can validate the contents of both $CH_{j,i}^r$ and $\{SH_i^{r+1}\}_q$ upon receipt of $PP_{i,q}^r$ from P_i.

But, to get a valid signature to convert $eCH_{j,i}^r$ to $CH_{j,i}^r$, P_i had to send at least t_{r-1} valid $PP_{i,q}^{r-1}$ messages to P_q, $q = 1, \ldots, t_{r-1}$ in round $r-1$ in order to get at least t_{r-1} $PP_{q,i}^{r-1}$ messages containing shares to derive K_i^r (and t_{r-1} unpolluted and validated chunks). This forces peers to relay chunks.

Because shares obtained in round $r-1$ are used to derive a given key k^r in round r, the following question arises:

Question 3.3. *How new peers in round* r *can relay a valid message?*

Answer: The splitter sends

$$SP_{j,i}^r = \{CH_{j,i}^r,\ SH_i^{r+1}\}, \tag{3.15}$$

(see Equations. 3.14 and 3.11) instead of $eSP_{j,i}^r$ (see Equation 3.9) for a peer P_i which is new in round r. Therefore, new HPs can send a valid PP message (see Equation 3.13) and get at least t_{r+1} shares to derive K_i^{r+1} in round r.

Notice that the threshold t_r used at the beginning of the round r to generate $A_{(n_r \times n_r)}^{r+1}$ to derive a given k_i^{r+1} by a peer P_i during round r fulfills Equation 3.8 under the assumption in Equation 3.7, if #MP$_r$ and #HP$_r$ do not change during round r. HPs perform a polite leaving, i.e. they inform to S about their leaving during round r and they actually leave after completing their round r. Therefore, only MPs can leave in a no polite way. This does not affect to HPs in deriving their k^{r+1} key due to Equation 3.8 but the chunk from S to MPs leaving after receiving it by S is lost. If this is detected, P2PSP activates the lost chunk recovery system mentioned in Chapter 2.

Algs. 3.3 and 3.4 show the pseudocode for the splitter (TP-SSS:S) and a peer (TP-SSS:P_i) of the new proposal using TPs and SSS, respectively. For the sake of simplicity, only the pseudocode for a HP P_i that is not new in round r has been taken into account in Alg. 3.4 (see Question 3.3).

Following we describe these algorithms.

- TP-SSS:S (using Trusted Peers and Shamir's Secret Sharing - Splitter's algorithm):
  - S generates a random secret key per chunk and peer in vector $K_{(nr)}^{r+1}$ which will be used to encrypt/decrypt the signature of future messages in round $r+1$, and a matrix $A_{(n_r x n_r)}^{r+1}$ of shares, n_r shares for n_r peers (see Line 4).
  - For each peer, the splitter sends a message as described in Equation 3.9 (see Line 8). Note that if P_i is a new peer, the splitter sends the message in Equation 3.15 instead.

Algorithm 3.3 TP-SSS:S.

```
Require: $L$, $S_{pr}$, $S_{pu}$
 1: $j \leftarrow 0$; $r \leftarrow 0$
 2: while $S$ is alive do
 3:     $t \leftarrow \lceil \frac{n_r}{2} \rceil$                #peers joining and leaving are taken into account
 4:     Generate $K^{r+1}_{(n_r)}$ and $A^{r+1}_{(n_r \times n_r)}$
 5:     for $i \leftarrow 0$ to $n_r$ do
 6:         $C_j \leftarrow$ getNextChunk()                #next chunk from source
 7:         $j \leftarrow j + 1$
 8:         sendMessage($eSP^r_{j,i}$,$P_i$)                #message to peer $P_i \in L$
 9:     if TP complains about $P_x$ and lost chunks then
10:         expelMP($P_x$)                #$L = L - \{P_x\}$
11:         recoverLostChunks()
12:     $r \leftarrow r + 1$
```

The expelling and lost chunk recovering mechanisms are the same as the used in Chapter 2.

- Algorithm TP-SSS:P_i (using Trusted Peers and Shamir's Secret Sharing - Peer's algorithm):
  - Upon the reception of a message $eSP^r_{j,i}$ from the splitter, P_i:
    * tries to reconstruct the secret key K^r_i using shares in $A^r_{i,*}$ (see Line 7).
    * decrypts the signature in $eCH^r_{j,i}$ and stores it in $CH^r_{j,i}$, in case K^r_i is correct (see Line 8). $CH^r_{j,i}$ will be relayed to the rest of peers in an *avoid-congestion mode*.
    * VerSing() procedure verifies the message signature. If the message from the splitter has correct signatures (see Line 9), P_i must finish round $r - 1$ and start round r. Therefore, it sends pending chunks in burst mode (see Line 10), instead of continuing using the *avoid-congestion mode*. The received and verified chunk C_j is sent to the player (see Line 12) and the share for me (P_i) and round $r + 1$ sent by S is stored (see Line 13).

Algorithm 3.4 TP-SSS:P_i.

```
Require: $S$
 1: $(L, S_{pu}) \leftarrow$ reveciveFrom($S$)
 2: while player is alive do
 3:     $(m, \text{sender}) \leftarrow$ receiveMessage()                 #wait for a message
 4:     if sender $==$ $S$ then                         #message from the splitter to me=$P_i$
 5:         $(eCH^r_{j,i}, SH^{r+1}_i) \leftarrow$ unpack($m$)                       #Eq. 3.9
 6:         $(C_j, j, P_i, r, E_{k^r_i}(\text{sign})) \leftarrow$ unpack($eCH^r_{j,i}$)          #Eq. 3.10
 7:         $K^r_i \leftarrow$ getMySecretKey($A^r_{i,*}$)
 8:         $CH^r_{j,i} \leftarrow \{C_j, j, P_i, r, D_{K^r_i}(E_{k^r_i}(\text{sign}))\}$
 9:         if VerSign($CH^r_{j,i}$) and VerSign($SH^{r+1}_i$) then
10:             sendBurstMode($PP^{r-1}_{i,y}$)                              #$y = q, \dots, P_{|L|}$.
11:             $q \leftarrow 1$                                   #next destination peer is $P_q$.
12:             sendToPlayer($C_j$)                                    #chunk from $S$ to me.
13:             $A^{r+1}_{i,i} \leftarrow \{SH^{r+1}_i\}_i$                           #store the share for me
14:     else
15:         $(CH^r_{j,x}, \{SH^{r+1}_x\}_i) \leftarrow$ unpack($m$)                  #from $P_x$. Eq. 3.13
16:         if VerSign($CH^r_{j,x}$) and VerSign($\{SH^{r+1}_x\}_i$) then
17:             sendToPlayer($C_j \in CH^r_{j,x}$)                            #chunk from $P_x$.
18:             $A^{r+1}_{i,x} \leftarrow \{SH^{r+1}_x\}_i$                  #increase the $r$+1 set of shares
19:             $PP^r_{i,q} \leftarrow \{CH^r_{j,i}, \{SH^{r+1}_i\}_q\}$
20:             sendMessage($PP^r_{i,q}$,$P_q$)                              #message to $P_q \in L$
21:             $q \leftarrow q + 1$
22:         else
23:             removePeer($P_x$)                                       #$L = L - \{P_x\}$
24:             if I am a TP then
25:                 sendComplaint($P_x$)                          #complains about the sender
```

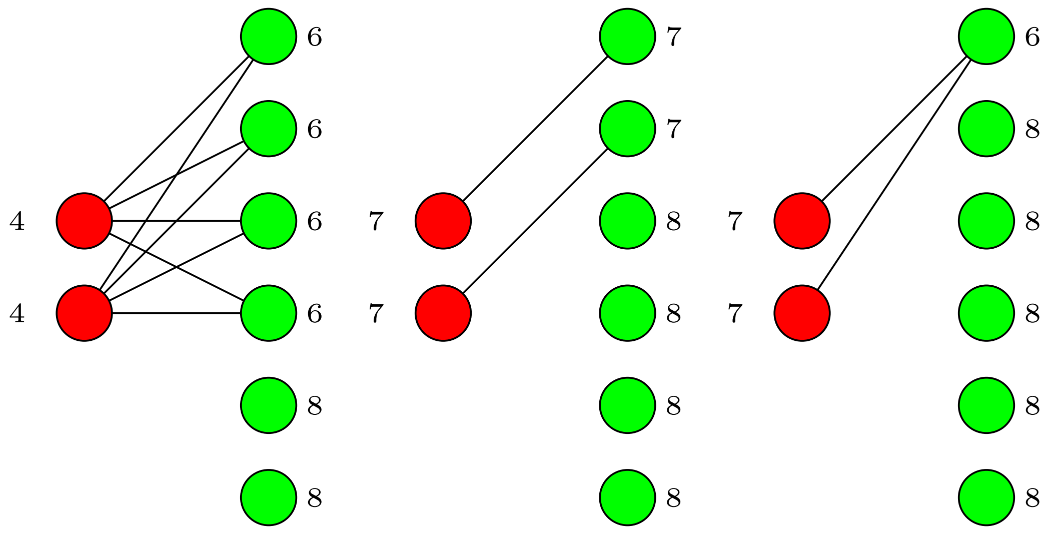
- Upon the reception of a message $PP^r_{x,i}$ from peer P_x, P_i:
  - verifies the signatures of the messages (see Line 16). If both signatures are right,
    - the chunk C_j is sent to the player (see Line 17).
    - the share $\{SH^{r+1}_x\}_i$ is added to the list of shares $A^{r+1}_{i,x}$, because it will be needed in round $r+1$ (see Line 7).
    - A message $PP^r_{i,q}$ including $CH^r_{j,i}$ and the share $\{SH^{r+1}_i\}_p$ for destination peer P_q is generated and sent to P_q (see Line 20).
  - If one of the signatures is incorrect, P_x is removed from L (see Line 23), and if P_i is a TP, it will complain about P_x (see Line 25).

The *spoofing* attack is one of the drawbacks of the shown proposal because an MP can impersonate an HPs and send polluted messages. There exist solutions for it, as the addition of a signature to any message, using the private key of the sender. This is not tackled here for the sake of simplicity.

The main idea of the proposal is to limit the number of HPs being attacked. In case all MPs collude, an MP could survive into the team only if it behaves correctly with $t_r - \#\text{MPs}_r$ HPs. So, it could attack to $\#\text{HPs}_r - t_r + \#\text{MPs}_r$ but no more. In case an MP attacks more than $\#\text{HPs}_r - t_r + \#\text{MPs}_r$ peers, it will never receive the necessary shares to reconstruct its keys for next rounds, under the Equation 3.8 assumption. Without keys, the only available options are: (1) to send a corrupted signature or (2) not to send anything. Therefore, a trusted peer will detect the MP, and it will be expelled from the team after a TP reports the attack to S.

Regarding the loss of quality in the received multimedia stream, the important factor is the number of chunks each HP lost, i.e., the number of MPs attacking to each HP. The worst case under Equation 3.8 is $t_r = \lceil \frac{n_r}{2} \rceil$ and $\#\text{MPs}_r = t_r - 1$. So, each MP can attack $\#\text{HPs}_r - 1$. This is only possible if there exist just one TP and it is the one not been attacked. It seems difficult to achieve it for the set of MPs. In such hypothetical case and having a large n_r, the collaborative attack may result in HPs cannot watch the streaming. A way to reduce the number of MPs attacking is to increase t_r.

MPs know the number of HPs and the value of t_r after having t_r shares with a positive verification of the S signature.



(a) One MP to many. (b) One MP to one. (c) Many MPs to one.

Figure 3.5: Three examples of possible pollution attacks. Circles are peers. Green ones are HPs and red ones MPs. Links among them show attacks and number beside a peer shows the number of obtained shares.

As we see, t is the most important parameter for the success of our proposal. Therefore, a strategy to establish the optimal value for t in Equation 3.8 could be developed. Let us show it with an example:

Having $n_r = 8$ and #MPs$_r = 2$ then #HPs$_r = 6$ and $t_r = 4$. MPs can attack up to four HPs. In the next round, S could decide to check $t_{r+1} = 5$. In such case and having the same team, if all MPs have attacked at least four HPs in round r, they will not be able to reconstruct k^{r+1} because they need three shares from HPs and only two HPs continue having good relations with them. However, HPs are able to generate their keys because they are six. Therefore, it is not appealing for MPs to attack to #HPs$_r - t_r +$ #MPs$_r$ peers (see Figure 3.5a).

In the previous example, S could select t_{r+1} in the range $[4, 8]$ without knowing #HPs. Using $t_{r+1} >$ #HPs$_{r+1}$ may produce the following undesirable results:

- One MP to many HPs (see Figure 3.5a). Using $t_{r+1} \geq 7$, not only the two MPs but also four out of six HPs cannot derive their keys for round $r + 2$.

- One MP to one HP (see Figure 3.5b). If $t_{r+1} = 7$, all peers can reconstruct their keys for round $r + 2$. However, if $t_{r+1} = 8$, two MPs and two HPs cannot do it.

- Many MPs to one HP (see Figure 3.5c). Using $t_{r+1} = 7$, an HP cannot derive its key, and the two MPs continue in the team. S cannot obtain evidence to determine who the attacker is (are). If $t_{r+1} = 8$ two MPs and one HP cannot derive their keys.

Notice that the most severe attack is the one shown in Figure 3.5a, in which it is easier to expel MPs. To expel MPs performing attacks shown in Figs. 3.5b and 3.5c are more difficult, but the impact of those attacks is not so hard. In those cases, establishing $t_{r+1} =$ #HPs$_{r+1}$ does not allow to expel MPs. Moreover, attacks in Figs. 3.5b and 3.5c are not appealing for MPs interested in breaking down the team quickly, taking into account that they could attack to a TP.

It would be interesting to devise a heuristic that expels peers with a low probability of them being HPs, based on the tracking of their behavior in the team.

Unfortunately, if the Equation 3.8 assumption is not fulfilled, MPs could have good behavior until they are more than t to carry out the attack.

3.4.2 Conclusions

In this study, we analyzed existing solutions to fight against free-riding and pollution attacks. Incentive mechanisms seem to be the best option for pull-based and non-fully connected overlays. We deal with a fully connected push-based scheme and propose a novel defense mechanism to force the peers to collaborate with the rest of the team based on the threshold of SSS. After a theoretical analysis of the proposal, we conclude that the most severe possible attack is fully mitigated. For the remaining attacks (although the impact is low), we can improve effectiveness to face them increasing the number of TPs. We advance some ideas about the need for an adaptive algorithm to establish the SSS threshold. The development of such algorithm, as well as an experimental analysis of the proposal will be addressed in a future work.

Chapter

4

Security in the P2PSP II: Software Defined Networks

As seen in the previous chapter, pollution attacks have a tremendous impact on push-based fully connected overlays in which each peer receives an exclusive chunk from the source, and it is also the only one responsible for relaying it to the rest of the peers. In this chapter, we propose a novel technique to identify and expel MPs which involves using TPs, Software Defined Networking (SDN) and proactive Moving Target Defense (MTD). Experiments to obtain the accuracy and effectiveness of the implemented methods, as well as an analysis of the performance concerns, were carried out through simulation using a Mininet network emulator. The experiments demonstrate the feasibility of our proposal, which provides high rates of detection, not only in pure SDN environments, but also in mixed ones.

4.1 Introduction

As a consequence of the static nature of the network, the current rules governing traditional network infrastructures make the detection and expulsion of MPs a challenge which must be addressed. A new architecture, called SDN, aiming to facilitate network management as well as enable efficient programming of network configuration is becoming more and more popular every day. In this context, the programmable feature of SDN makes it possible to dynamically modify the destination endpoint in the packets to implement a moving target defense. For this reason, we propose a novel technique based on SDN to identify and expel MPs from the overlay in a short period of time, avoiding incorrect decisions, i.e., without expelling honest peers.

The main contributions of this chapter are the following:

- To propose an SDN-based MTD approach to mitigate the MP attacking problem in P2P.
- To propose a method for detecting and expelling MPs in a P2P push-based fully connected overlay in a deterministic way, avoiding *false positive* and *false negative* results observed in other methods such as peer polling, or trust management systems.
- To perform experiments and analyze the performance of the proposed technique using a simulator of P2PSP.
- To study the feasibility of applying our proposal to a mixed network using SDN and non-SDN devices.

A review of the related work in targeted attacks and MTD topics is shown in Section 4.2. Section 4.3 describes our proposal for discovering and expelling malicious peers by using an SDN approach. Several experiments are carried out in Section 4.4 to obtain the accuracy and effectiveness of the proposal, used in conjunction with P2PSP. Finally, our conclusions are shown in Section 4.5.

4.2 Related work on Moving Target Defense

Recent research on the proactive defense concept has led to two major mechanisms: (i) MTD, which increases the complexity, diversity, and randomness of the cyber systems to disrupt adversaries' reconnaissance and attack planning, and (ii) cyber deception, which provides plausible-looking yet misleading information to deceive attackers [74]. MTD is a research hotspot in the field of network security [1]. A survey on MTD can be found in [77]. Our approach can be classified as Moving Target Network Defense (MTND) with IP address and port mutation [78]. Network Address Space Randomization (NASR) force hosts to change their IP addresses frequently [3]. Instead of performing a random change of peers' IPs, a transparent moving target defense using SDN is presented in [31]. In a similar way, our model is based on a random change of target IPs among peers by scrambling peers' EPs in SDN devices. In this way, the original P2PSP is unaltered. An introduction to network address shuffling can be found in [7].

4.3 Studied solution

Keeping our previous strategy for combatting pollution/free-riding attacks nearly intact by using TPs [53] (see Chapter 3), we introduce an SDN controller algorithm which is applied only to the UDP traffic involved in the P2PSP protocol.

4.3.1 Detection method

As shown in Chapter 3, the main weakness of adding TPs in a P2P team is that TPs could be discovered by MPs and, as a result, they could perform targeted attacks, thereby avoiding the pollution of chunks whose destination is a TP. Therefore, a simple way to prevent such as situation is hiding the destination EPs from the entire team (excluding the splitter). However, in a P2P system, every peer needs to communicate with the rest of the peers directly, so they have to send packets to their specific EPs. To achieve the goal of changing the destination EPs, we have designed and implemented the pseudocode shown in Algorithm 4.1[1]. This algorithm is located in the SDN controller as a Ryu app [64], and consists of exchanging the destination EPs of messages sent by peers using an association table. The entries of this table are indexed by the destination EP to which a peer intends to send a message, and they contain the actual EP of the peer that will receive it. An example is presented as follows (see Figure 4.1):

1. The splitter S sends a chunk to P_1, the first peer in the team. The OF (Open Flow) switch redirects the packet to the Ryu controller as it has no rule in charge of managing this. The controller takes a reactive approach, i.e., it analyzes all messages from S to know when a new round will arise (a round starts when S sends a message to the first peer in the list of peers in the team).

[1]The full version of the code is available as open source at `https://github.com/P2PSP/SDN-P2P`.

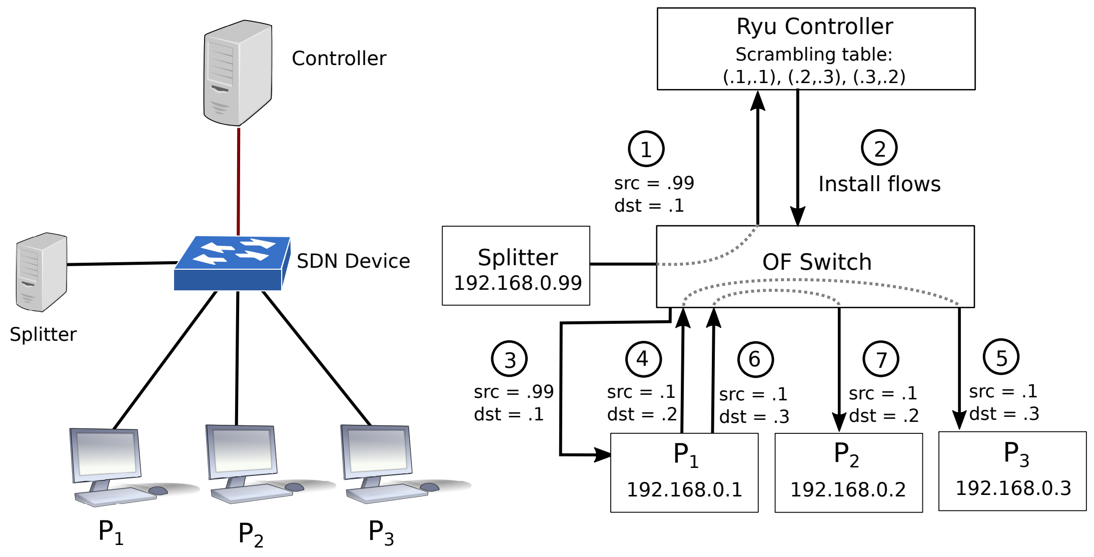


Figure 4.1: A simple version of the network architecture with only one SDN device. The communication scheme is also shown. The ports of the EPs have been ignored in this example for the sake of simplicity. Note that source (src) and destination (dst) IPs are represented in the figure with the last octet in decimal format.

2. The Ryu controller, which runs our P2PSP Ryu app (Algorithm 4.1), generates a table, the so-called *scrambling table*, which associates the original and actual destination EPs of the peers. This assignment is randomized in each round. The scrambling table is sent in the form of rules (allowed flows) to the SDN device using the OpenFlow protocol [45].

3. After installing the new flows, the chunk is sent to the destination P_1.

4. P_1 sends a chunk to the first peer in its list of peers. Let us assume that it is P_2. Therefore, the source of the message is the IP address 192.168.0.1, and the destination IP is 192.168.0.2.[1]

5. When the message reaches the SDN device, the destination is changed and then redirected to the new EP. In the example of the Figure 4.1, P_3 (192.168.0.3) is

[1]In this discussion, the ports of the EPs have been ignored for the sake of simplicity.

the actual destination address, and P_3 receives the message.

6. P_1 sends a chunk to the second peer in its list of peers, which is P_3 according to our example. Therefore, the source of the message is the IP address 192.168.0.1, and the destination IP is 192.168.0.3.

7. As in Step 5, when the message reaches the SDN device, it redirects the message following the rules received from the controller. Then, the original destination is changed to a different destination. In this case, P_2 (192.168.0.2) is the destination which receives the message.

The random shuffle applied to the EPs of the peers in the team can cause the new destination to match the source of the message. In this situation, the controller makes a new change to send the message to another destination different from that of the source. For instance, according to the scrambling table in Figure 4.1, when P_2 sends a message to P_3, the new destination is P_2 ((.3,.2) in the scrambling table), but this would cause the loss of the message. In this case, the controller uses the element in the scrambling table that has the original origin (.2), which is (.2,.3). Thus, the final destination is P_3. This does not cause duplicates at reception.

4.3.2 Vulnerability analysis

Since peers can only decide to send the chunks or not and, in the cases where they are sent, whether they are polluted or not, it is impossible for MPs to bypass the system using these attacks.

In the specific case of the P2PSP protocol, one of the rules in DBS says that "peers stop communicating with a peer if they do not receive anything from it". When using an SDN, this rule constitutes a weakness because an attacker could know the real peer who received its packet. For instance, if an MP attacks peer P_i, and in the next round P_j does not send anything to the MP, it could know that the EP assigned in its list of peers for P_i in the previous round was actually mapped to the EP of P_j. A similar procedure could be used by a group of MPs to detect whether they are behind an SDN device. With the aim of making MPs unable to perform target predictions, we opted to remove said rule from the protocol and each HP continues forwarding chunks to all members of the team, even when it is attacked.

Algorithm 4.1 P2PSP Ryu app algorithm.
INPUT: peer_list, packet, splitter

```
 1: scrambling_table = dict(zip(peer_list, peer_list))      #Create a scrambling table for
    assosiating original EPs with new destinations EPs
 2: if P2PSP UDP packet then
 3:    if splitter's packet and new round then
 4:       clean_flows()                                     #Remove rules from devices
 5:       shuffle(scrambling_table)                         #Assign a new EP mapping
 6:    else if packet destination in scrambling_table then
 7:       store mapping for IP to Mac and Mac to Port
 8:       if scrambling_table[destination] is myself then
 9:          destination = myself
10:       new_destination = scrambling_table[destination]
11:       add_Flow(new_destination)                         #Add new rules for the device
12: send_PacketOut()                                        #Send modified packet
```

We could imagine a challenging scenario in which the number of MPs is half of the team, and each MP attacks just one peer in a collaborative way (in which attackers share information about the attack). Thus, they can know the current scrambled EP for the TPs: which are the EPs used by the expelled peers. However, as soon as a new random scrambling is done, all TPs are hidden again.

To determine the viability of our approach we evaluate the ~~following~~ security MTD properties of our system. For this purpose, we use the properties listed in [25, 43]:

- Unpredictability: As previously stated, our approach includes the use of a random number generator to shuffle the EPs in a transparent way for peers. Thus, MPs cannot guess the real destination EP by themselves in a new scrambled round.
- Vastness: The scrambling is perfomed on a current set of peers' EPs. Exhausting the EP (or even the IP) space is a general attack that will not affect the P2P protocol for the current set of peers.
- Periodicity: The scrambling period parameter guarantees the periodicity.

- Uniqueness: Although peers do not know who the real destination is, the controller knows this information all along. Therefore, all peers in the team are uniquely identified by their real EPs.
- Revocability: All flows in our system are specific to the destination. So, it is possible to modify individual records.
- Availability: Our approach does not introduce new availability constraints and thus meets the availability property.
- Distinguishability: Trusted peers are in charge of distinguishing HPs from MPs and expelling the latter.

4.3.3 **Probability of detection**

Let us suppose we are dealing with a team of 10 peers, in which one is a TP and one is an MP in a managed network, and the scrambling is performed before every round. MP only pollutes one chunk or does not send a chunk in each round. The probability that the MP attacks the TP in the first round is $p = \frac{1}{9} = 0.111$. However, what is the probability of TP being attacked in the second round? For that to happen, two events must occur: i) The MP does not attack the TP in the first round, and ii) the MP does attack the TP in the second round. Therefore, the probability is $p = \frac{8}{9} \times \frac{1}{9} = 0.098$. Based on this information, the probability that the MP attacks the TP in round r in a team of size $Z > 2$ is

$$p = \left(\frac{Z-2}{Z-1}\right)^{(r-1)} \times \frac{1}{Z-1}. \tag{4.1}$$

However, we are more interested in knowing the probability of detecting an MP after N rounds than precisely in round r. Therefore, the expression of the probability we are interested in is

$$p = \sum_{r=1}^{N} \left(\left(\frac{Z-2}{Z-1}\right)^{(r-1)} \times \frac{1}{Z-1} \right). \tag{4.2}$$

4.3.4 **Cost of performing scrambling**

There is certain essential information that must be known in order to measure the impact of implementing this solution in an SDN environment. It could be defined in terms of network traffic, such as the exchange of messages between the controller and switches, or the number of flows to be installed. It could also be defined in terms of detection cost. Both options will depend on how many rounds of scrambling take place, yet this will be different in each scenario.

As for the network cost, this can be reduced by decreasing the number of rounds in which a new scrambling takes place. However, this action increases the detection time because the higher the number of rounds to redo the scrambling, the later the detection of the MPs.

Therefore, the additional costs added by this solution are:

- Traffic of chunks from S to the controller in order to monitor the round status (caused by a reactive approach). It could be optimized by sending direct orders from S to the controller only in each new round (proactive behavior).
- Flows cleaning and re-installation, which will depend on the scrambling period, one or several rounds.

Every time that a scrambling is performed, three related tasks are required, which in turn affect the overhead:

1. Cleaning the flows in all SDN devices. To do so, the controller has to send a *OFPT_FLOW_MOD* message [60] to all devices in order to remove every flow from their tables. The process of sending such a packet entails an overhead of 48 bytes for each SDN device.
2. Scrambling EPs in the controller dictionary. This involves a cost related to the CPU usage, which will depend on the number of EPs to scramble and, of course, on the used algorithm used to generate randomness. This overhead can be considered negligible for present-day SDN controllers.
3. Installing new flows in all SDN devices. A new *OFPT_FLOW_MOD* packet is sent to each SDN device where peers are connected. It is also necessary to send flows to create connections among SDN devices. Therefore, the overhead in this step is the sum of computing the following expression for each SDN device in our managed network: 48 bytes $\times$ (connected peers $+$ SDN connected devices).

4.3.4.1 **Modifying the scrambling period**

One of the main concerns in our approach is related to the increase in the network traffic and the number of flows installed in the SDN devices. Both of them are closely related to the scrambling period parameter.

The probabilities of detection are reduced as the scrambling period is increased. These results can be modelled by adding the scrambling period parameter s to the expression in Equation 4.2, resulting in

$$p' = \sum_{r=1}^{N} \left(\left(\frac{Z-2}{Z-1} \right)^{(\lceil \frac{r}{s} \rceil - 1)} \times \frac{0^{(r \mod s)}}{Z-1} \right). \tag{4.3}$$

In this way, the network manager can decide on a trade-off between the scrambling period r and detection efficiency p'.

4.3.4.2 **Minimizing costs**

In the current implementation, we take an *SDN reactive* approach to managing flow entries installation, which means that if there is not a match for the flow in the SDN device, it creates a *packet-in* packet [60] and sends it to the controller for further instructions. The controller analyzes the *packet-in* and sends the appropriate flow entries to the device. Therefore, if the scrambling period is one (changes in each round), the flow entries are valid only for one round and have to be updated for each round. In cases where the audience is known beforehand, for instance in pre-paid pay-per-view events, we could take an *SDN proactive* approach. Rather than reacting to a packet, it consists of populating the flow tables from the controller to all SDN devices for any future traffic that matches the flows among all possible peers. If some predicted peers are not connected, this solution only affects the overhead because peers send chunks to non-existent peers, but the condition that all existing peers receive the stream is fulfilled. Moreover, we could adopt a number of controller location techniques for scalability and traffic optimization reasons [34].

4.3.5 **Mixed network using SDN and non-SDN devices**

Our approach was designed for proper operation in a network architecture where all devices are equipped with SDN technology. However, it could prove to be of great use under a network which mixes traditional and SDN devices. To accomplish this, some decisions must be made about where the scrambling is performed.

Assuming a network with SDN and non-SDN devices we must consider different scenarios:

- SDN devices are in the core of the networks, i.e., no peer is connected directly to them. Under this scenario, it is very risky to carry out an EP scrambling because we must first know the routing table for every device in the network which is traversed by chunks in order to make sure that all peers receive all chunks. Unfortunately, this is something that cannot be guaranteed.

- SDN devices are the gateways of a set of peers (see Figure 4.2). In this case, the scrambling can be performed in the outgoing chunks, but also in the incoming chunks if the destination EP is also connected to a SDN device. Since the controller is a centralized entity, it takes control of the scrambling by analyzing the new packets and sending the appropriate flows to the respective devices.

In order to achieve the goal of applying the studied solution to mixed environments, we have added a minimal update to the P2PSP Ryu app. It consists of adding a list of devices managed by the controller and informs on which peers are connected to each managed device. With this modification, we can verify whether a message has been previously modified by the controller or not. The idea is to avoid a double EP scrambling and to make sure that all peers receive all chunks once.

For the sake of simplicity, let us assume that there is a team with $Z = \{Z_{in} \cup Z_{out}\}$ peers. One of them is an MP and another is a TP. The managed network has Z_{in} peers, including the TP. So, Z_{out} peers are outside the managed network. We can differentiate four possible attacking scenarios:

1. **Attacks happen outside the managed network:** When an attack occurs between peers in Z_{out}, the attack cannot be detected because the scrambling algorithm is not working among these peers and there is no outside TP. However, this attack does not affect peers in Z_{in}.

2. **Attacks happen inside the managed network:** Attacks among peers in Z_{in} will be detected thanks to the scrambling approach, using $Z = Z_{in}$ in Equation 4.2 or Equation 4.3.

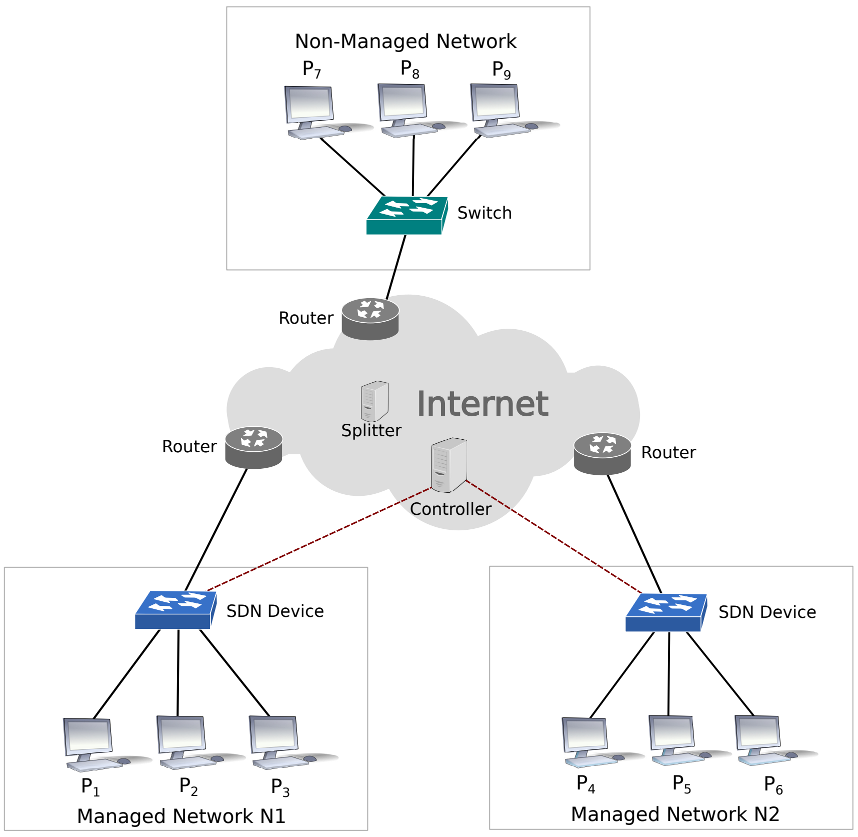


Figure 4.2: An example of a network architecture with SDN and non-SDN devices running the P2PSP with our SDN approach.

3. Attacks from outside to inside the managed network: If a peer in Z_{in} is attacked by the MP in Z_{out}, the attack is detected because the scrambling can change the destination peer to another peer in the managed network, that could be the TP. In this case, the probability of detection is given by using $Z = Z_{in} + 1$ in Equation 4.2 or Equation 4.3, i.e., the number of peers inside the managed network plus the MP. Equations are correct only if the MP attacks the managed network in all rounds.

4. Attacks from inside to outside a managed network: When the MP in Z_{in} attacks a peer in Z_{out}, the attack is not detected. Although the attack does not cause damage in the SDN environment, it affects peer in Z_{out}. To avoid this, we could improve the proposed algorithm by using two different scrambling tables:

a) A list with peers in Z_{in}, which is used for scrambling when the chunks come from outside.

b) A list with all peers, which is used for scrambling when the chunks leave the managed network.

The pseudo code for the Ryu controller including this improvement is shown in Algorithm 4.2.

Therefore, the probability of detection (see Equation 4.2 and Equation 4.3) is reduced by $\frac{Z_{in}}{Z}$, which is the probability of redirecting the MP's message to the managed network.

Previous considerations can be extended to instances with several managed networks where SDN devices are controlled by the same controller, and the scrambling table contains the Z_{in} peers in all managed networks. Consequently, the studied solution works in a mixed network maintaining some of the benefits as in a fully SDN architecture.

4.4 Experiments

In all the simulations shown in this section peers are in a managed network. Simulations were carried out with Mininet network simulation tool [36], on an Intel Core i7 CPU 860 @ 2.80GHz x 8 machine with 16 GB of RAM, using Ryu SDN Framework as the SDN Controller [64].

4.4.1 Fixed vs variable targets attacks

Assuming EPs are shuffled in each round, an MP could think that changing the target could reduce the probabilities of being detected, especially if the scrambling is pseudo-random, meaning a totally different EP association is forced for each round. In that case, using a variable target could prove helpful because, if the target is fixed after n rounds, it means the MP has attacked the entire team. However, the same does

Algorithm 4.2 P2PSP Ryu app algorithm for mixed networks.
INPUT: peer_list_b, packet, splitter

```
1: peer_list_a = remove_external_peers(peer_list_b) #Remove peers outside the management
   networks
2: scrambling_table_outside=dict(zip(peer_list_a, peer_list_a)) #Create a scrambling table
   assosiating original EPs with new destinations EPs for using among peers outside (only includes
   peers into the management networks)
3: scrambling_table_inside=dict(zip(peer_list_b, peer_list_b))                    #Create
   a dictionary for assosiating original EPs with new destinations EPs for using among peers inside
   (it includes all peers into the team)
4: if P2PSP UDP packet then
5:    if splitter's packet and new round then
6:       clean_flows()                                  #Remove rules from devices
7:       shuffle(scrambling_table_outside)              #Assign a new EP mapping
8:       shuffle(scrambling_table_inside)               #Assign a new EP mapping
9:    else
10:      if packet comes from outside then
11:         scrambling_table = scrambling_table_outside
12:      else
13:         scrambling_table = scrambling_table_inside
14:      if packet destination in scrambling_table then
15:         store mapping for IP to Mac and Mac to Port
16:         if scrambling_table[destination] is myself then
17:            destination = myself
18:         new_destination = scrambling_table[destination]
19:         add_Flow(new_destination)                   #Add new rules for the devices
20: send_PacketOut()                                    #Send modified packet
```

not occur in the case of a totally random scrambling because the EP mapping is unpredictable. For this reason, we decided to implement a totally random mapping. In this scenario, the probabilities of an MP being detected should be the same regardless of the attack method, whether it be fixed or variable.

We run a set of experiments with the following parameters:

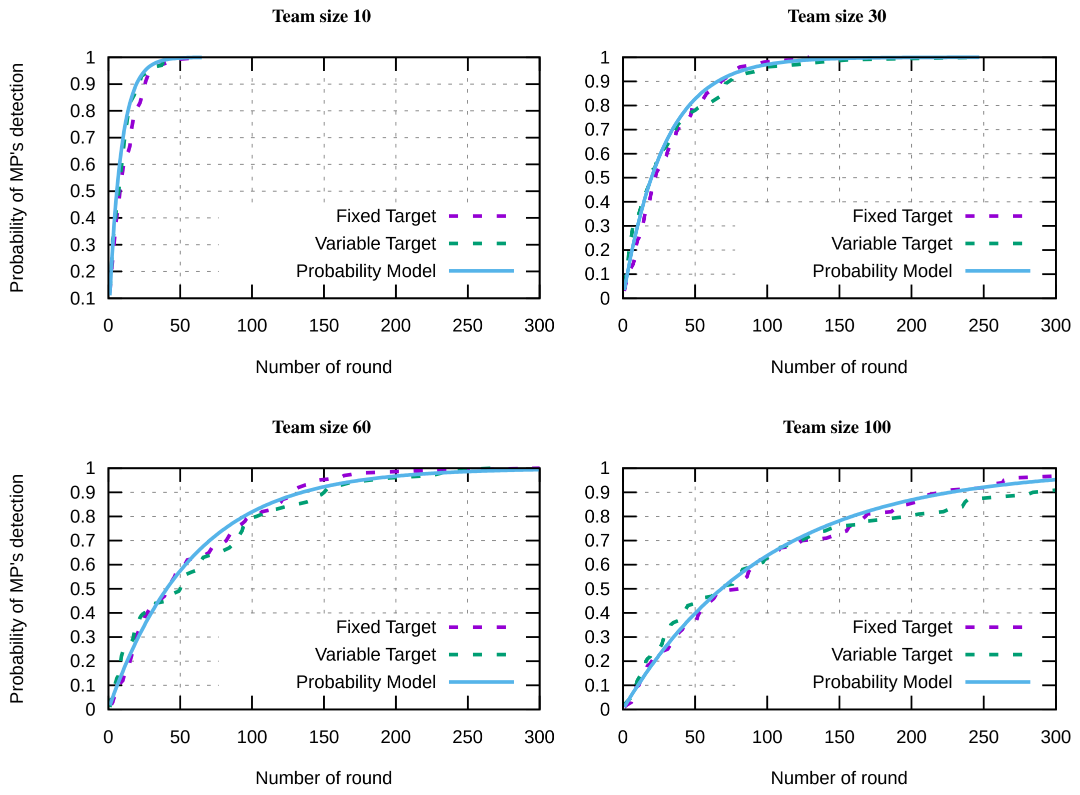


Figure 4.3: Experimental detection probabilities of an MP performing one attack per round. Scrambling period is one.

- Team size: from 10 to 100.
- Samples: 100 samples (executions) for each instance.
- Period: 1, i.e., all the executions perform one scrambling per round.
- Cases: fixed and variable target attacks. One per round.

Results of the experiments are shown in Figure 4.3. For both fixed target and variable target attacks the results fit quite well with the probability model p in Equation 4.2. An MP is usually detected before 60 percent of the number of rounds equals the team size. This percentage is the same as the average percentage of vulnerable computers contacted as the normalized shuffle rate increases, which was calculated in

[8]. Whereas the perfect shuffling cost results in a drawback due to the relatively high percentage for an attacker to find a vulnerable host, in our case the scrambling is worth its cost due to the relatively high percentage to reach the trusted peer.

This justifies the idea of keeping the EP mapping totally random in order to avoid a smart attack where MPs can figure out how the dynamic scrambling is carried out for a hypothetical pseudo-random system. For all team size configurations, the probability of detection is very close to one when the number of rounds triples the size of the team. For instance, in a team with 100 peers, the probability of detection before round 300 is 0.95. Thus, assuming an average bitrate of 2500 kbps (typical HD 720p video), and a chunk size of 1024 bytes, the MP will be detected and expelled before $t = 98.3$ seconds.

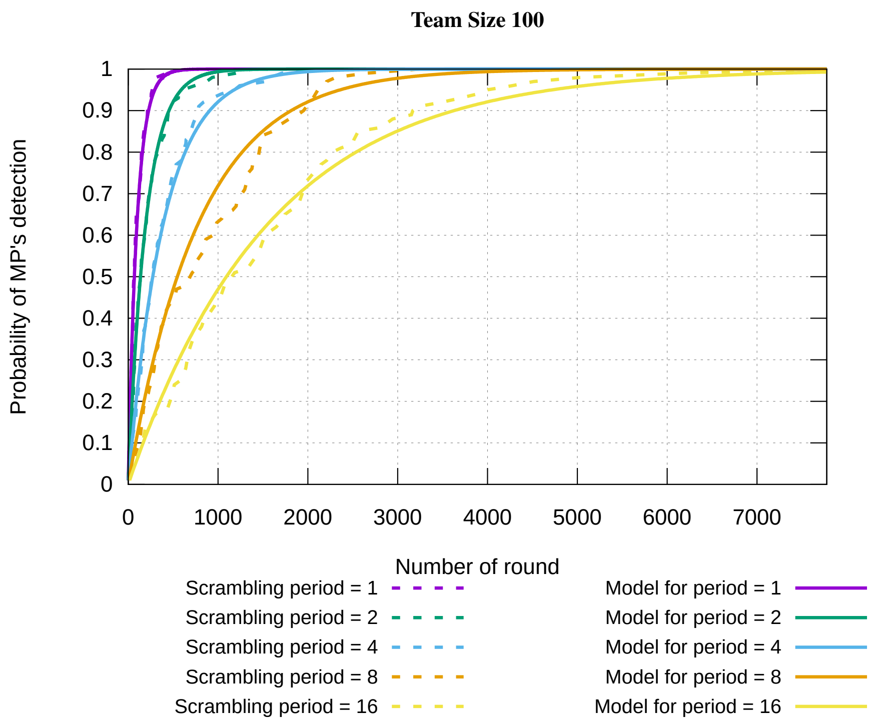


Figure 4.4: Experimental results for MP detection probabilities with different scrambling periods.

Regarding the consequences for the detection system when opting for a fixed or variable type of attack, we conclude that the probabilities of detection are quite similar. Therefore, an MP will be detected and expelled in the same way regardless of the type of attack it carries out. Additionally, a variable attack makes no sense in a traditional network without scrambling because after each target change is made, the MP is nearer to being discovered. Moreover, MPs performing more than one attack per round will increase their probability of being detected.

4.4.2 Modifying the scrambling period

Modifying the scrambling period parameter helps us to reduce the CPU usage, network traffic, and the number of flows installed in devices. However, it causes a loss in the efficiency of MP detection. Figure 4.4 shows a set of experiments carried out for a team of 100 peers varying the scrambling period (100 samples for each one).

The simulated MP detection is close to the probability model shown in Equation 4.3. As expected, the number of average rounds to detect the MP with a probability greater than 0.9 increases significantly with the scrambling period.

4.5 Conclusions and future work

In this chapter, we have analyzed a moving target defense based on scrambling the target EPs in SDN devices in order to detect malicious peers in peer-to-peer fully connected push-based overlays by using trusted peers. The experiments have demonstrated the viability of our proposal, resulting in a relatively quick MP detection (in relation to the overlay size), not only in pure SDN environments but also in mixed environments where some peers are on the Internet and others are under managed networks.

Since modifying the scrambling period has a severe impact on the performance of the detection algorithm, we propose optimizations such as a proactive approach and controller location techniques as possible future lines of work.

Chapter
5

Improving Efficiency on P2PSP Through NAT Traversal

Network Address Translation (NAT) [67] is commonly used to solve the lack of public addresses. However, devices performing NAT also hinder the communication between processes in the internal network hosts and those in external networks. This is a problem in P2P networks where peers need to know each others before establishing a communication. Some protocols, such as STUN [42] and ICE [62], have been developed to cope with this problem. They use public IP addresses and port discovery, and UDP Hole Punching (UHP) [68] to address the traversal in connection independent mapping NATs. However, some NAT devices perform connection dependent mappings, causing different public ports to the same process in the internal network to have connections with different endpoints. In this case, port prediction is needed by UHP. Indeed, a number of port prediction techniques have been developed previously [56, 70, 75]. Herein, we propose a new and simple NAT Traversal algorithm that uses Collaborative Port Prediction (NT-CPP), which provides an effective NAT traversal for all combinations of port assignment and endpoints filtering strategies, as long as the NAT behaviour can be predicted.

5.1 Introduction

In current P2P systems, most peers run behind devices performing NAT. As depicted in RFC2663, a device performing NAT (typically a home router) interconnects two different networks, one internal (in most cases, a "private" network) and one external (generally, a "public" network). A NAT device maps bidirectional correspondences between internal and external networked entities using a collection of *NAT translation table entries*. These correspondences express connections between external and internal endpoints. A connection is determined by two EPs (Endpoint$_1$,Endpoint$_2$). An EP is represented by M:M^i where M is the IP address, and M^i is the port used by the process to communicate. For the sake of simplicity, we will use M^i to denote the EP instead of M:M^i.

The way peers establish connections between them does not depend on whether they are in the internal or the external network. Therefore, in that sense, NAT is transparent for peers.

NAT is performed mainly for two reasons. First, NAT considerably reduces the number of IP addresses needed by Internet users because all the processes that are behind a NAT device can be seen as they are using the external IP address of the NAT device when they communicate with other external processes. Second, NAT devices reduce the chances of establishing connections from the external networks to the internal ones, which can increase the level of security in the processes running in the internal network. More specifically, a NAT (device) allows incoming (external-to-internal) traffic only if there is an entry in the table that explicitly enables it.

As has been described in RFC4787, there are different types of algorithms for assigning the external ports used by NATs. These can be classified as [32]:

- *External Endpoint Independent Mapping (EEIM)*: EEIM NATs reuse the same (external) port s for all traffic sent from the same internal EP X to any external EP Y.

- *External Endpoint Dependent Mapping (EEDM)*: EEDM NATs use a different (external) port $s(Y)$ for a different external EP Y.

- *Connection Dependent Mapping (CDM)*: In this case, the assigned (external) port $s(X, Y)$ is changed for each connection (X, Y). CDM NATs can increase the index assigned to the new port: (1) using a Predictable (PCDM) port step Δ, or (2) Randomly (RCDM).

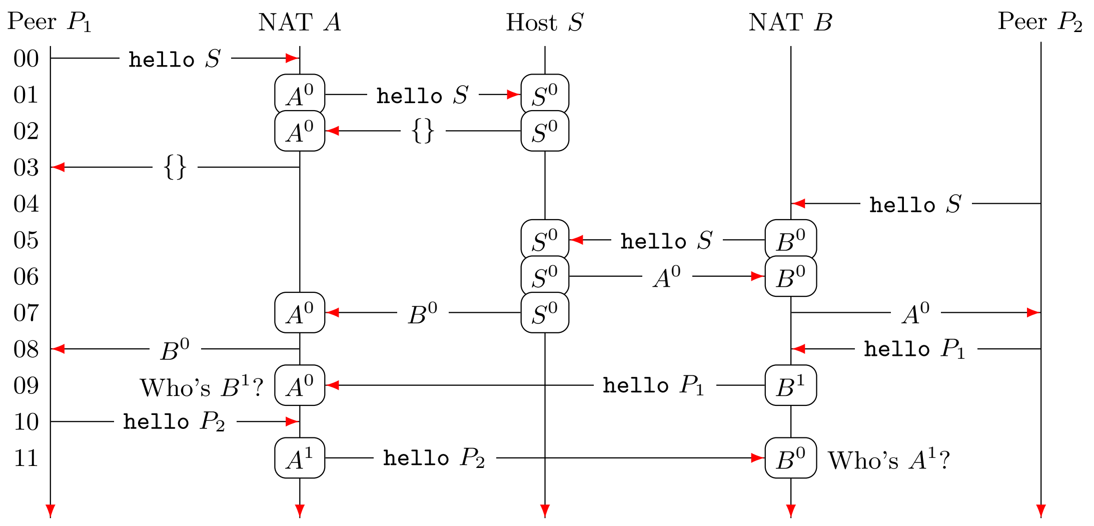


Figure 5.1: This example shows the most common case with two "private" peers P_1 and P_2, behind CDM NATs. Even with the help of a "public" host S, peers cannot communicate with each other. More specifically, what happens is that NAT device A does not have a translation entry for endpoint B^1, and NAT device B does not have a translation entry for EP A^1.

Also, as has been described in RFC4787, depending on how the incoming traffic (from the external to the internal network) is filtered, NATs can be classified as [32]:

- *Endpoint Independent Filtering (EIF)*: All incoming packets are forwarded.
- *Address Restricted Filtering (ARF)*: Only incoming packets from the same external IP address to which the initial outgoing packet was sent to are forwarded.
- *Endpoint Restricted Filtering (ERF)*: Only packets that come from the same external EP that the initial outgoing packet was sent to are forwarded.

To better understand the problem that we want solve, Figure 5.1 shows an example of a connection failure between two NATed peers P_1 and P_2 that are behind CDM NAT devices, when port discovery based protocols (that do not perform port prediction), such as STUN (RFC5389) are used [28]. As can be seen in Step 11 of the interaction, NAT device B only relays to the internal network the incoming traffic from host S (a

Peer1/2	EEIM/EIF	EEDM/ARF	PCDM/ERF	RCDM/ERF
EEIM/EIF	yes	yes	yes	yes
EEDM/ARF	yes	yes	*(yes)*	no
PCDM/ERF	yes	*(yes)*	*(yes)*	no
RCDM/ERF	yes	no	no	no

Table 5.1: Theoretical traversal success for different NAT mapping and Endpoint filtering combinations.

similar situation happens at NAT device A), which makes it impossible to establish a direct connection between the peers.

Instead of using a simple port discovery, some port prediction could be performed by P_2 to determine that the external port that NAT A will use to send the data from P_1 to P_2 is A^1, and P_1 to determine that the external port that NAT B will use to send data from P_2 towards P_1 is B^1. A set of ports, instead of only one, can be predicted in order to be used by UHP [55] to increase the probability of establishing a connection between the peers.

To foresee also the limits of our proposal, we must also consider that the implicit dynamism in the creation of EPs in internal networks makes such prediction difficult for some combinations of NAT types. Table 5.1 summarizes the expected success of NAT traversal techniques based on UHP. The cells labelled as "*yes*" refer to those NAT combinations where NT-CPP should work using only one predicted port. The cells marked "*(yes)*" represent those combinations where NT-CPP with a relatively small number of predicted ports results in peers' connection. The cells marked "no" represent those combinations in which the NATed peers establish a communication only by using an unlimited number of port predictions.

The remainder of this document is organized as follows. In Section 5.2, an analysis of existing solutions for CDM NAT traversal is shown. Section 5.3 describes *NAT Traversal using Collaborative Port Prediction* (NT-CPP), which constitutes our contribution. Next, in Section 5.4 the performance of NT-CPP is evaluated. Finally, the main conclusions and future work are summarized in Section 5.5.

5.2 Related work on CDN NAT traversal

One of the first solutions to provide CDN NAT traversal was proposed by Takeda [70]. This approach is based on the use of a single public STUN server [63] with two different IP addresses, which peers must contact in order to determine the external EPs used by the NAT devices involved. Once peers have been contacted twice with the STUN server, it sends to peers the collected information about the other peers. Then, peers use an "allocation discovery process" to build a ports prediction. After that, an invitation packet is sent towards the other peers they want to communicate with. The method can be repeated with different destination ports in order to increase the chances of a successful connection. Unfortunately, [70] does not define port prediction techniques, nor does it report experimental results.

Another traversal approach was developed by Wei et al. in [75]. In this case, two public servers are used to compute the behaviour of NAT devices. When an incoming NATed peer P_j (echo server) wants to communicate with another, already monitored by a server, NATed peer P_i (echo client), P_j contacts the public servers and a "punching mode" computed for P_i's NAT device. The public servers also send P_i a message showing that P_j wants to communicate with it, and the punching mode for the P_j's NAT device. Then, P_j sends a number N_p (1000 in the reported experiments) of low-TTL UDP packets towards a predicted external port at P_i's NAT device, in order to create N_p translation entries in P_j's NAT device pointing to P_i's external port. Next, P_i sends N_p high-TTL UDP packets towards the P_j's predicted external port. Finally, if at least one of these packets reaches P_j, both peers can communicate. Although very high success ratios were reported, the experiments were performed with just two NATed peers P_j and P_i. In most of the cases, there exist several NATed peers in an internal network. These conditions make it more difficult to achieve a successful prediction. Additionally, sending 1000 packets has some drawbacks: a large number of NAT table entries for each internal peer, the required time, and it can be treated by the NAT device as a port scanning, resulting in blacklisted peers.

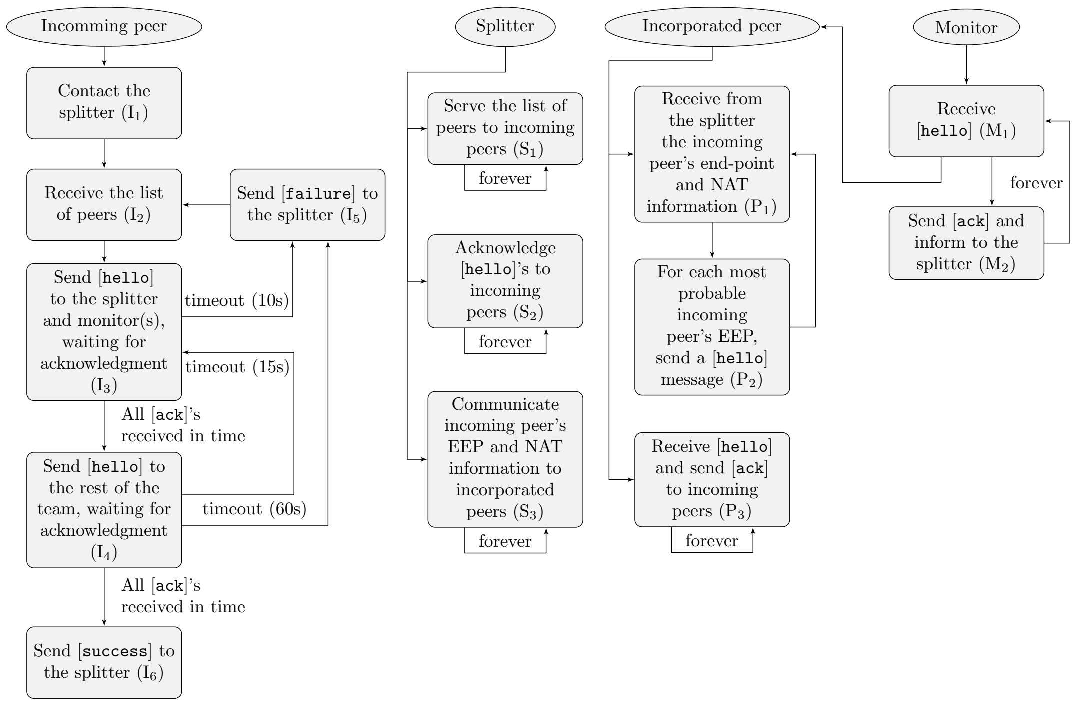

Incomming peer
Contact the splitter (I1)
Receive the list of peers (I2)
Send [failure] to the splitter (I5)
Send [hello] to the splitter and monitor(s), waiting for acknowledgment (I3)
timeout (10s)
timeout (15s)
All [ack]'s received in time
Send [hello] to the rest of the team, waiting for acknowledgment (I4)
timeout (60s)
All [ack]'s received in time
Send [success] to the splitter (I6)
Splitter
Serve the list of peers to incoming peers (S1)
forever
Acknowledge [hello]'s to incoming peers (S2)
forever
Communicate incoming peer's EEP and NAT information to incorporated peers (S3)
forever
Incorporated peer
Receive from the splitter the incoming peer's end-point and NAT information (P1)
For each most probable incoming peer's EEP, send a [hello] message (P2)
Receive [hello] and send [ack] to incoming peers (P3)
forever
Monitor
Receive [hello] (M1)
forever
Send [ack] and inform to the splitter (M2)

Figure 5.2: States diagrams of the entities involved in the NT-CPP UDP handshake.

5.3 NAT Traversal using Collaborative Port Prediction

5.3.1 Handshake

As can be seen in Figure 5.1, the public server S could predict (and serve) the ports A^1 and B^1 to the peers P_2 and P_1 in the Steps 06 and 07, allowing them to establish a communication. NT-CPP is based on a similar procedure, and on the idea that increasing the number of public servers, and therefore the number od used ports, can increase the chance of a correct port prediction.

NT-CPP is supposed to be used in P2P systems, such as browser-to-browser video conferencing [21] or P2PSP [52]. In the case of P2PSP, an incoming peer, after contacting with the *splitter* (the public server of a P2PSP *team*), will transmit [hello] messages successfully to at least N *monitor* peers, which can also act as public servers. Monitor peers will report NATing information about the incoming peer to the rest of the team. As will be shown in the experiments, higher N values increase the probability of traversing NATs with a small number of predicted ports. Thus, little traffic is generated at low resources consumption throughout the UHP stage. For the sake of simplicity, and without loss of generality, the rest of the document will be framed within P2PSP (see Chapter 2).

NT-CPP defines a set of steps executed by all team members (see Figure 5.2), which are triggered when a new peer P_j is joining the team. First, P_j must contact (using TCP) the splitter S (see box I$_1$). After that, S sends (using TCP) the current list of peers to P_j (see boxes S$_1$ and I$_2$), and appends P_j at the end of its list. In order to create a translation entry in P_j's NAT device for the incoming UDP traffic, P_j sends (throughout the reception of the list) UDP [`hello`] messages to S and peers in the list of peers. Each [`hello`] message is acknowledged with an [`ack`] or a number of failures (timeouts) are produced, in which case the incorporation of P_j is aborted (see I$_5$). On the contrary, P_j tells S that its incorporation has been successful (I$_6$), and previously incorporated peers perform a port prediction using the information received from S about the NAT translation information collected from P_j's NAT (see P$_1$). Monitors behave as normal peers, except that they are contacted by P_j first.

Figure 5.3 shows a detailed example of how NT-CPP works in ideal conditions[1]. The splitter S and monitor M use public ports S^0 and M^0, respectively. When two new NATed peers, P_1 and P_2, arrive at the team, the following events occur:

00. M requests to join the team (the joining request is not shown in the figure for brevity), and S sends M an empty list of peers. At this moment, M has joined the team.

01. P_1 requests S to join through an external port A^0 (again, this message is not shown). S sends the list of peers to P_1, which is a sequence of tuples (end-point, port-step, peer index). At this moment, this list contains only the endpoint M^0 (in the figure, this information is represented with the tuple $\{(M^0, 0, \#M)\}$ where $\#M$ is the index of M in the list of peers of S).

02. NAT device A relays towards P_1 the previous message.

03. P_1 sends a [`hello` M] towards M.

04. A relays the previous message, which is received by M. Due to the fact that A is a CDM NAT device, a new external EP A^1 is used for this message.

[1]When the first predicted port at both ends is correct. Otherwise, peers will try more predicted ports.

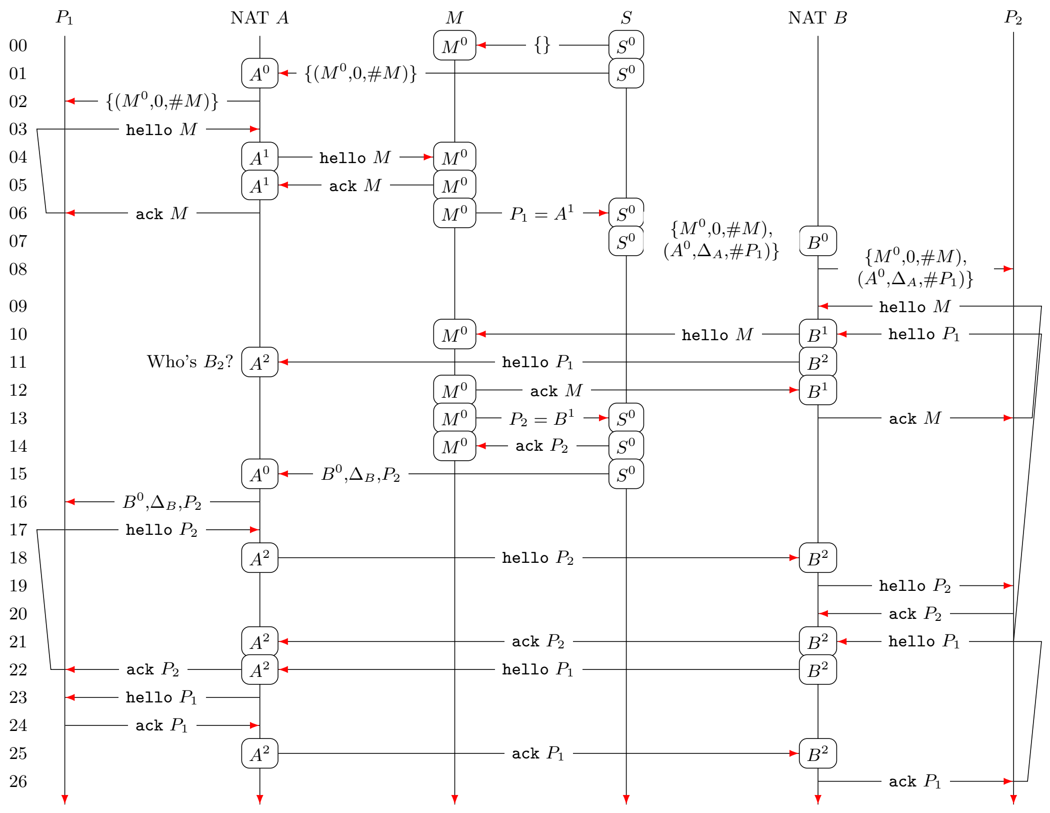


Figure 5.3: Timeline of an (ideal) UHP interaction between two peers P_1 and P_2 which are behind CDM/ERF NATs.

05. M sends [`ack` M] from M^0 towards A^1.

06. The previous message is relayed by A to P_1. Simultaneously, M informs S that P_1 has communicated with it, using the external EP A^1.

07. P_2 requests to join the team (not shown) and S sends the current list of peers to it, which contains $\{(M^0, \Delta_M = 0, \#M), (A^0, \Delta_A, \#P_1)\}$, where A^0 is the external EP used by P_1 to communicate with S, and Δ_A is the port step calculated for NAT device A, measured by S. port prediction for the external port that A should assign to P_1, when P_1 initiates communication with P_2. This prediction results in a set of ports (see Equation 5.3).

08. B retransmits the previous message to P_2.

09. P_2 sends a [`hello` M] towards M.
10. B retransmits the previous message using the external EP B^1, which reaches M, and P_2 sends a [`hello` P_1] towards the predicted port A^2 in Step 07.
11. The previous message reaches A^2 and it is discarded because there is no working entry for the connection (B^2, A^2) in the translation table yet.
12. M acknowledges the [`hello` M] received in Step 10 toward B^1.
13. The [`ack` M] message is received by P_2 and M informs S that P_2 is also using the external EP B^1. S uses the information about P_2 external EPs to compute the maximum port step Δ_B in the NAT device B.
14. S acknowledges the previous reception.
15. S sends the message [(B^0,Δ_B,S^1)] to P_1. B^0 is the external EP used by P_2 to talk with S. Δ_B is the maximum port step measured for P_2.
16. P_1 receives the previous message.
17. P_1 sends [`hello` P_2] to external EP B^2.
18. NAT device A relays the message [`hello` P_2] towards B.
19. NAT device B relays the message [`hello` P_2] towards P_2 At this moment, P_2 knows that P_1 is able to send data to it.
20. P_2 acknowledges the [`hello`] messages.
21. [`ack` P_2] is received by NAT device A and the timer assigned to the message [`hello` P_1] sent in Step 10 timeouts. Therefore, this message is re-sent to A^2.
22. P_1 receives [`ack` P_2] and NAT device A receives [`hello` P_1].
23. [`hello` P_1] is delivered to P_1. At this moment, P_1 knows that P_2 is able to send data to it.
24. P_1 acknowledges the previous [`hello` P_1].
25. [`ack` $\mathtt{P}_1$] arrives to NAT device B.
26. [`ack` $\mathtt{P}_1$] reaches P_2.

5.3.2 Characterization of NAT algorithms

As mentioned, an incoming peer P_j sends [hello] messages to the splitter and monitor peers. Monitors are always placed in the first positions of the list of peers with a fixed ordering. Let $\{A^0, A^1, \cdots, A^N\}$ the external ports used by P_j's NAT device A to send the [hello] UDP packets towards S and the N monitor peers, in a sorted order. Let's denote

$$\Delta_i = A^i - A^{i-1},\ i = 1, 2, \ldots, N \tag{5.1}$$

the port distances gathered by S. Then, a *port step* for P_j's NAT device A is computed as

$$\Delta = \begin{cases} 0 & \text{, if } \forall i, \Delta_i = 0 \\ \text{GCD}(\Delta_1, \cdots, \Delta_N) & \text{, otherwise} \end{cases} \tag{5.2}$$

where GCD is the Greatest Common Divisor operator. Some considerations have to be taken into account:

1. Normally, peers share the local network with other entities and therefore, a number of UDP packets could be sent through the P_j's NAT device by these entities over the course of the P_j incorporation. In this case, it is possible that some external ports are skipped. Hopefully, the higher the number of monitors, the smaller the impact of this problem, because the probability of skipping a port between each initial connection to one of the monitors decreases.

2. Δ is the highest (maximum) possible port step. This detail will be taken into account in the port prediction.

NT-CPP classifies a NAT algorithm depending on the value of Δ:

1. $\Delta = 0$: The NAT device is using the same external port for communicating P_j with the other peers in the team. Then, P_j is behind an EEIM NAT. Public (not NATed) peers are also considered to be using this type of NAT.

2. $\Delta > 0$: The NAT uses a different external port for each external peer. Then, P_j is behind a CDM NAT. In this case:

   (a) If $\Delta_1 = \Delta_2 = \cdots = \Delta_N$, P_j's NAT device is using PCDM.

   (b) If $\Delta = \lim_{N\to\infty} \text{GCD}(\Delta_1, \cdots, \Delta_N) = 1$, P_j's NAT device uses RCDM.

5.3.3 Ports prediction algorithm

A ports prediction is performed by each incorporated peer P_i after receiving from S a message indicating that the incorporating peer P_j wants to join the team (see, for example, Step 15 of Figure 5.3), and also for the incorporating peer P_j. This message contains (for NAT device B):

1. The external EP (B^0) that P_j (P_2) opens in its NAT device (B) to communicate with the public server (S).
2. The port step (Δ_B), which will determine the rest of the ports tried by the incorporated peers P_i.

The output of the ports prediction algorithm is a sorted list of ports. The port prediction was designed considering that:

1. S and all the peers of the team (including P_j) have exactly the same list of peers, apart from itself.
2. Although Δ ports were skipped during the communication between P_j and the splitter/monitors, this could be due to non-P_j's traffic or to an unpredictable NAT behavior. In the first case, the predicted ports should be selected sequentially (as in [75]). However, in the second case, the prediction should use a step Δ. This is the main difference between [75] and our proposal. As the experimental results will show, the consideration of a $\Delta > 0$ improves the efficiency of the prediction by minimizing the number of UHP steps to achieve a successful P_j incorporation.

Taking into account previous considerations, the list of port predictions that a peer P_i performs for a peer P_j behind NAT B is determined by:

$$\begin{array}{rcl} \{s(P_j)\} & = & B^0 + j + \{s \in \{0, 1, \cdots, n/2 - 1\}\}; \\ \{s(P_j)\} & += & B^0 + (j + \{s \in \{n/2, 1, \cdots, n-1\}\}) \cdot \Delta_B. \end{array} \tag{5.3}$$

where "$+=$" denotes the concatenation of lists and n is the number of guessed ports, B^0 is the first external port that P_j used to communicate with S (through NAT B), and Δ_B is the port step measured for P_j's NAT B.

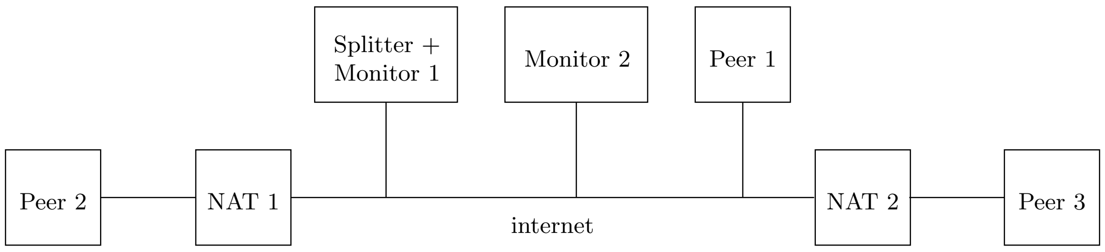


Figure 5.4: Simulated network used in tests. Each box is a host running Arch Linux, including the NATs that are implemented with `iptables`.

5.4 Evaluation

The performance of NT-CPP was evaluated in the simulated environment described in Figure 5.4, where two PCDM NATed peers, P_2 and P_3, try to establish a connection (using NT-CPP) with the other peers in the presence of *alien traffic* (generated by other processes not showed in the figure), that also allocates external ports in the NAT devices. As Figure 5.4 shows, one of the monitor peers runs on the same host as the splitter to minimize the latency and the loss of packets.

The configuration scripts for this network setup use `iptables`[1] and *Linux network namespaces*[2]. With the aim of simulating a real network, a delay (30 ms), a jitter (5 ms) and an average rate of packet loss (1 %) are configured for each network interface.

The port skips generated at the NATs by the P2P and the alien traffic were modeled using an exponential distribution with a rate parameter λ. In order to provide statistical significance to our results, each experiment will be carried out 100 times, and the rate of successful connections (the port prediction success rate) will be provided.

5.4.1 Comparison between different port prediction algorithms

As Equation 5.3 shows, NT-CPP expresses a compromise between two extreme alternatives: (i) as in [75], the use a sequential port prediction (generated by NT-CPP if the measured port step were $\Delta = 1$), and (ii) the use of the measured port step Δ

[1] https://github.com/P2PSP/core/tree/master/doc/P2PSP/NTS/iptables
[2] https://github.com/P2PSP/p2psp-console/blob/master/src/setup_NAT_network.sh

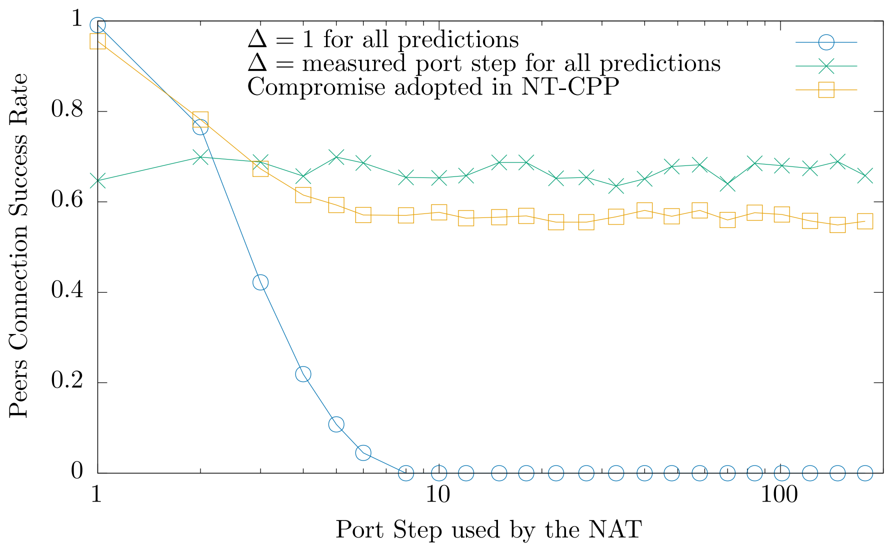


Figure 5.5: Performance of the proposed ports prediction algorithm (NT-CPP) compared to two extreme cases: (i) predict all the ports using $\Delta = 1$ for all the predictions (notice that this behaviour can be obtained if we force $\Delta = 1$), and (ii) predict all the ports using the measured port step ($\Delta =$ measured port step).

for generating all the predicted ports. Figure 5.5 shows the performance of the three alternatives: i) port step one, represented by the line with circles, ii) port step Δ, represented by the line with X's, and iii) NT-CPP schema which predicts half of the ports using $\Delta = 1$ and the other half using the measured Δ, represented by the line with squares. The port step used by the NAT was varied between 1 and 200. As can be seen in the Figure, the compromise adopted in NT-CPP shows a good average performance because it is suitable for low and high port step values. For this experiment:

1. Only $N = 1$ monitor peer was used.
2. The rate parameter used for the exponential distribution to model the traffic in the NAT devices was $\lambda = 2.0$.
3. $n = 20$ port predictions were performed.

5.4.2 Impact of the number of monitors

For this experiment, the network described in Figure 5.4 was simulated using the following parameters:

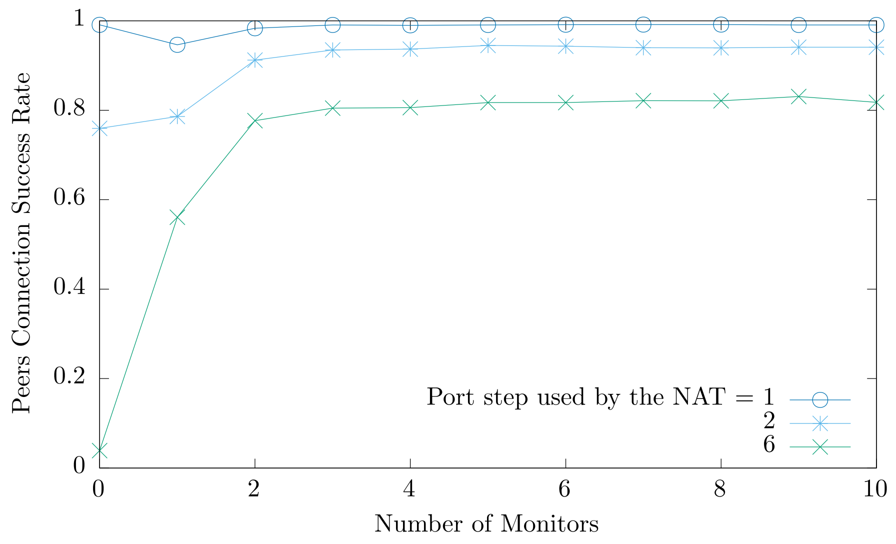


Figure 5.6: Impact of number of monitor peers (N) in the rate of successful connections as a function of the (measured) port step (Δ).

1. The number of monitors was varied between $N = 0$ and $N = 10$.
2. The port step used by the NATs was varied between 1 and 6.
3. The rate parameter of the exponential distributions modeling the port skips in the NAT devices was $\lambda = 2.0$.
4. $n = 20$ ports were predicted using NT-CPP.

The success rate for this experiment is displayed in Figure 5.6. Notice that only 20 ports were predicted and the success rate is over 0.8, which is a much smaller number of predicted ports than the 1000 ports used in [75] for obtaining similar success rates (one monitor is equivalent to the configuration in [75]). Due to NAT ports are randomly assigned, there is not enough port correlation to take advantage of using more than two peers. Therefore, It can be observed that with 2 monitors the performance is close to the maximum possible with NT-CPP.

5.4.3 Impact of the port step used in the NAT devices

The network described in the Figure 5.4 was simulated using the following parameters:

1. The number of monitors was varied between $N = 0$ and $N = 2$.

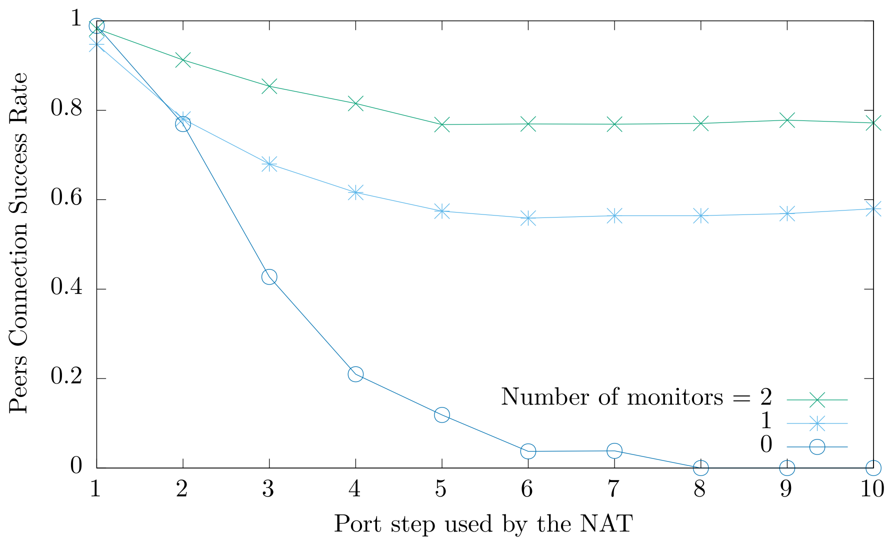


Figure 5.7: Impact of the port step used in the NATs in the rate of successful connections, depending on the number of monitor peers (N).

2. The port step used by the NAT devices was varied between 1 and 10.
3. The rate parameter of the exponential distributions modelling the port skips in the NATs was $\lambda = 2.0$.
4. $n = 20$ ports were predicted using NT-CPP.

The success rate for this experiment is displayed in Figure 5.7. As expected, the higher the port step used by the NATs, the lower the port prediction success ratio. However, it remains stable when the port step is higher than 5.

5.4.4 Impact of the number of predicted ports

Figure 5.8 shows the performance of the NT-CPP for different combinations of the number of predicted ports, the port step used by the NATs, and the number of monitors. Again:

1. The number of monitors was varied between $N = 0$ and $N = 2$.
2. The port step used by the NATs was varied between 1 and 2.
3. The rate parameter of the exponential distributions modeling the port skips in the NATs was $\lambda = 2.0$.

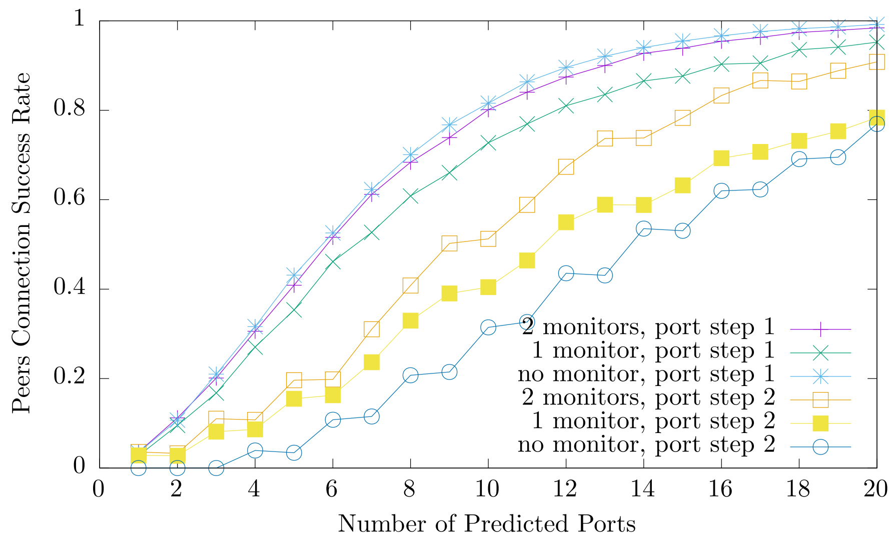


Figure 5.8: Impact of the number of predicted ports (n) in the rate of successful connections, depending on the number of monitor peers (N), the port step used by the NAT devices, and the number of predicted ports.

4. From $n = 1$ to $n = 20$ ports were predicted using NT-CPP.

The prediction success rate increases significantly when the number of monitors is also increased. For instance, for a port step of 2 and 10 predicted ports, the success rate increases from 0.35 (no monitors) to 0.55 (two monitors). Notice that the port prediction success rate increases as the number of monitors also increases, except for the rare case in which the port step used by the NAT devices is always 1. In this case, the success rate is slightly lower using NT-CPP because in the extreme situation where $\Delta = 1$ for all the predictions, the optimal algorithm is to consider all the ports predictions sequentially, not only the first half of them (as NT-CPP does).

5.5 Conclusions

Collaborative port prediction can accurately classify the type of NAT and estimate a port step that makes it possible to generate a port prediction that, when used with UHP can help to establish connections between processes that are running behind those NAT devices. Experiment results have shown that only a low number of guessed source ports is needed for the traversal success. Only two monitor peers collaborating leads to higher traversal success rates, compared with solutions with only two public servers, as in [75], especially for port steps greater than one. In this case, and considering that this could incorporate security issues to the NAT traversal algorithm, the incorporation of the information provided by all the peers of the team could help to improve the prediction.

The proposed approach could be applied to web peer-to-peer systems, as a preliminary alternative to force the use of TURN servers to deal with CDM NAT devices. This aspect will be evaluated in a future work. Moreover, we are interested in applying this model in real life NAT devices, as well as adapting it to more complex NAT addressing schemes.

Chapter

6

Runing P2PSP on Embeded Devices

This chapter shows that P2P technology is suitable for broadcasting real-time events among low-end devices (such as smartphones and Google Chromecast). The aim is to avoid third-party media servers in order to decrease the cost of the streaming and increase the degree of privacy. Experimentation provides insights about memory and CPU requirements in such devices, and demonstrates the advantages of the P2P architecture compared to the client/server architecture in terms of performance.

6.1 Introduction

Nowadays, multimedia streaming systems are used massively. According to Cisco Systems, more than half of the data transmitted through the Internet is related to audio and video. This fact has fostered the implementation of different streaming architectures, which essentially enable broadcasting of multimedia streams to large audiences as efficiently as possible.

The most popular streaming architecture is based on the CS (Client/Server) model, where the streams are uploaded to a single server which broadcasts they to a set of users. The success of this architecture results from its simplicity, and full control of the transmission by the content provider. On the other hand, pure CS systems have some drawbacks, such as a poor scalability and privacy issues [17]. Scalability can be increased by using CDNs (Content Delivery Networks). However, P2P architectures are more scalable because the content provider sends only one or a small number of copies of the media, and peers dedicate their upload bandwidth to relay the media as many times as needed. Moreover, this turns P2P solutions more private, because the stream does not travel through any third-party intermediate nodes such as Youtube or Facebook Live servers.

Today, a large part of client devices in the multimedia streaming ecosystem are mobile. However, their small screens sometimes provide a poor QoE (Quality of Experience) to end users, especially for high-definition contents [15]. As a solution, Google released Google Chromecast (GC) to relay a stream from an auxiliary host to a local display. Here, we show how to run a P2PSP peer in GC, requiring a low computational burden to receive and render real-time Full-HD media. This idea is shown in Figure 6.1, where a mobile device broadcasts *only* a copy of the stream to three GC devices.

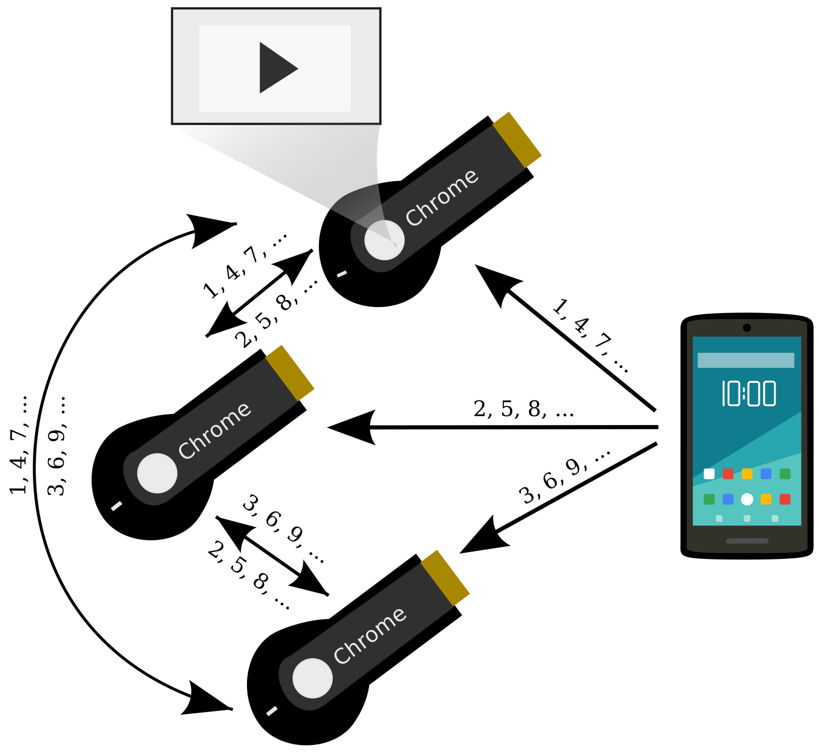


Figure 6.1: Streaming from a smartphone using a P2P approach without intermediate media servers.

Having in mind that the target devices have limited resources, e.g. small memory and/or low CPU performance, we integrate several Web technologies into the development of a P2P system for live-video streaming. Some of these technologies have been already released, such as HTML5 and JavaScript, but others such as P2PSP (Sec. 2), WebRTC (Sec. 6.4.3) and MSE (Sec. 6.4.3.4) are in their development process. Any WebRTC-powered Web browser can run previous technologies. The number of devices running them increases every day.

The main contributions of this study are: 1) a novel implementation of a P2P streaming protocol in GC and other WebRTC-based devices, 2) experiments showing that our proposal works smoothly on resource constrained devices, without third-party media servers, which increases the privacy, and 3) to show the appropriateness of a P2P model for media streaming in resource-constrained devices compared with a CS model.

6.2 Related work on P2P in resource-contrained enviroments

In recent years, the number of multimedia devices which use the Internet is increasing, some of them based on P2P systems. For example, the integration of a P2P protocol in an IPTV set-top box was studied in [40]. However, this solution needs to purchase a specific device and configure it, which complicates the deployment of the system and therefore reduces its popularity. We show here a solution where the installation/update of the software does not depend on the device because it just consists in downloading a Web page, which contains the JavaScript code for running a P2PSP peer. We use a generic GC as a resource-constrained device for the illustration.

Due to the limited resources in some electronic devices, some authors moved the P2P network to the cloud, where every device had a different access point to the media content [23]. The main advantage of this approach is that the users do not need to install or update the P2P-streaming software. Nevertheless, this also means that the bandwidth needed to run the P2P system is also required on the cloud side, which does not allow to scale as much as on a pure P2P system, where computation and transmission are moved to the end-user systems.

In [35], a resource analysis of live content streaming from mobile devices was done. There also exist P2P live video streaming applications for Android devices [22].

Currently, there are Android TVs, but most of the Android devices are smartphones, which offer a low QoE, mainly because their displays are too small.

Nowadays, Web browsers can run more complex codes, providing delivery and playback of multimedia content. JavaScript and HTML5 are able to run a wide range of applications. However, other emergent technologies, such as WebRTC, are needed to allow direct communication between browsers through a P2P connection. There are several applications of WebRTC for different contexts, such as videoconferencing [57] or Video on Demand [58]. In [48] we proposed to run a P2PSP peer in a Web browser.

6.3 Google Chromecast (GC)

GC is an HDMI device that extends the traditional capabilities of typical home televisions by providing them with an Internet connection and a high-definition multimedia player. 1st generation GCs incorporate a Marvel ARMADA 1500 Mini, and 512 MB of RAM memory. It also includes hardware for decoding VP8 and H.264 video compression formats. 2nd generation GCs use a Marvel ARMADA 1500 Mini Plus, 2.5 times faster than the 1st generation. It is also able to decode VP9. The latest version, GC Ultra, in addition, can manage HEVC at 4K-resolution.

Mainly due to its reduced price, GC has become a popular streaming platform with millions of users all over the world. Regarding the software, GC runs a tailored version of ChromeOS. The key feature of this work is the availability of a Chrome Web browser, which is compatible with HTML5, JavaScript, and WebRTC.

GC can be used in two different ways: 1) it receives content from another auxiliary device, such as a smartphone, a computer or a tablet, and 2) it downloads multimedia content directly from the Internet using its Chrome Web browser, following the instructions of the auxiliary device. Here, we show that GC is also able to run a P2P protocol.

6.4 Technologies used in this study

One of the most interesting facts about smartphones and GC is the inclusion of the Google Chrome Web browser in its set of applications. The constant evolution of Web browsers make them capable of executing complex codes instead of being just HTML renderers. Thanks to the evolution of HTML itself and other APIs, there exist numerous Web applications that resemble desktop applications such as multimedia players, complex finance software or modelling tools. In fact, Web applications are becoming more and more popular over desktop software. Some advantages are platform independence, no installation, and ease of use. However, there still exist limitations on the kind of applications that can migrate to the browser, as multithreaded ones.

6.4.1 HTML5 and JavaScript

HTML code is interpreted by Web browsers providing a multimedia interface between the user and a remote application. In HTML5 (HyperText Markup Language 5), new elements and attributes are established, reflecting the evolution of Web pages and multimedia technologies. For example, multimedia elements are introduced, such as `<video>` and `<audio>` tags, providing multimedia features through video and audio codecs directly at the browser through a properly standardized interface.

Additionally, JavaScript is commonly used to create dynamic Web applications and user interface enhancements on the client side. JavaScript can also be used on the server side as well as in standalone applications.

6.4.2 WebSocket

WebSocket is a protocol which provides bi-directional communication channels between Web servers and browsers but not between browsers. WebSocket allows establishing persistent connections, and both sides can start the information exchange at any moment. WebSocket was designed thinking in HTTP and HTTPS. Therefore, it can use ports 80 and 443. Other ports can also be used by exchanging HTTP messages. WebSocket is also supported by HTTP proxies. The WebSocket protocol was standardized as RFC 6455 in 2011 by the IEFT (Internet Engineering Task Force), and W3C (World Wide Web Consortium) is working on standardizing its API.

6.4.3 WebRTC

WebRTC (Web Real-Time Communications) is an open source project intended to become a new standard to extend the real-time communication capabilities of Web browsers. WebRTC allows real-time communication between Web browsers using a Peer-to-Peer architecture. W3C and the IETF are currently finishing the definition of this API. The goal is to accomplish a JavaScript API that, along with the required labels in HTML5, can define a new protocol to allow a direct communication between Web browsers. Moreover, the API defines other elements to access the stream of webcams and microphones, among other common peripherals, without the need to install additional plugins. WebRTC is already available in most commonly used Web browsers.

6.4.3.1 **PeerConnection**

It is a WebRTC component to manage a stable and efficient communication for data streaming between peers, i.e., a direct communication between Web browsers that are running an instance of the same JavaScript application. In order to know other peers and to agree on the parameters of the communication, exchange of messages is needed before achieving a peer connection. This is performed through a signaling server (see Section 6.4.3.3). Therefore, once the Web browsers (peers) know each other, they can communicate directly using a secure peer-to-peer connection without using an intermediary server, which increases the security of the transmitted data. In order to allow UDP data to traverse NAT devices, PeerConnection uses ICE (RFC 5245), along with STUN (RFC 5389) or TURN (RFC 5766). Only in the case of symmetric NATs, encrypted data is relayed using a TURN server, which does not follow the P2P philosophy.

6.4.3.2 **DataChannel**

Before the existence of WebRTC, sending data among browsers was an inefficient process, less scalable and with a lack of privacy, because a server was needed to relay the content to the Web browsers. Now, the *DataChannel* API of WebRTC provides a direct bidirectional data communication between peers, as well as a flexible set of data types.

DataChannel has two operating modes: the unreliable mode and the reliable mode. The unreliable mode has less overhead and offers a faster operation, but it ensures neither the delivery of the messages nor the order of their arrival. On the other hand, the reliable mode does ensure them but its higher overhead makes it less appropriate for live streaming.

6.4.3.3 **Signaling Server**

The WebRTC specification establishes that there must exist a server that lets a peer introduce itself to others. Therefore, when a new peer arrives, this intermediary server announces it to the rest of the team. WebRTC does not specify how the signaling must be carried out. There are many different options to deploy a signaling server. Some standard protocols already used for similar functions like SIP (RFC 3261) or XMPP (RFC 6120) may serve as a base for the deployment. However, other protocols can be used as shown in Sect. 6.5.

Using WebRTC, the contact among peers must be performed by a mandatory *signaling stage* which consists of following the Session Description Protocol (SDP) and an Offer-Answer state machine process.

6.4.3.4 **Media Source Extensions (MSE)**

The MSE is a specification of the W3C HTML Working Group with the goal of extending the functionalities of the *HTMLMediaElement* object. MSE allows developers to use JavaScript to build dynamic media streams with `<audio>` and `<video>` tags. *MediaSource* represents a multimedia data source for an *HTMLMediaElement* object. The *SourceBuffer* object is used to move multimedia data into the buffer at the same time that the video tag element is playing. The byte stream formats accepted by MSE specification are MP4, WebM, and MP2T.

6.5 Splitter and peers implementation

6.5.1 Splitter + Signaling Server

One of the most important entities in a P2PSP team is the splitter, since it is in charge of receiving the stream, splitting it into chunks, and sending them to the peers. The splitter also introduces new peers to the team, performing the same task as the signaling server in WebRTC. Therefore, it seems reasonable to merge both entities to save resources and to avoid redundant functionality. However, this is not mandatory. For instance, if we use a smartphone as a splitter, the signaling server usually keep in a separate public server.

To perform a full duplex communication between the Web browser and the signaling server, we use WebSockets over TLS (RFC 5246). WebSockets is a good choice for a signaling server to perform the introduction between peers because a browser compatible with WebRTC is usually also compatible with WebSockets. The use of JSON with WebSockets is appropriate to perform the exchange of the "session description" object in SDP used by WebRTC. To increase privacy and security, a password is requested to join the team. To increase reliability, redundant signaling servers could also be used.

6.5.2 Peer + player

In GC, P2PSP peers must also render the stream. The playing of the stream is performed using the MSE component. Therefore, as soon as the peer buffer is full, the streaming content is sent to the player by moving chunks into the `MediaSource` object buffer. Backward media seeking capability depends on the memory usage and services provided by the player which is out of P2PSP. Chunks are transmitted between WebRTC peers over `DataChannels`. This API relies on SCTP (RFC 4960), which in turn, runs over DTLS (RFC 6347) to cypher the data transfer, which is a mandatory requirement of WebRTC.

Summarizing, our HTML5 application runs on the Chrome browser. We use WebSockets to perform the communication with the signaling server. WebRTC receive video chunks as data. This data is sent with the video tag through MSE to render it.

6.6 Experimental results

The transmitted video consists of the first 120 seconds of Big Buck Bunny video (`peach.blender.org`), which is an MP4 (H.264 + AAC) fragmented at 24 FPS. Two resolutions were used: 640x360 and 1920x1080 pixels.

The experiments focus on comparing the computational resources required for the P2P and the CS models at the video server side and at the GC side. Since the resources needed by a P2P-peer or CS-Client at GC do not depend on where the rest of P2P-peers or CS-Clients run, all of them were run on a desktop PC, except the monitored P2P-peer and CS-client which were run on a GC. The open source code used is available at p2psp.org.

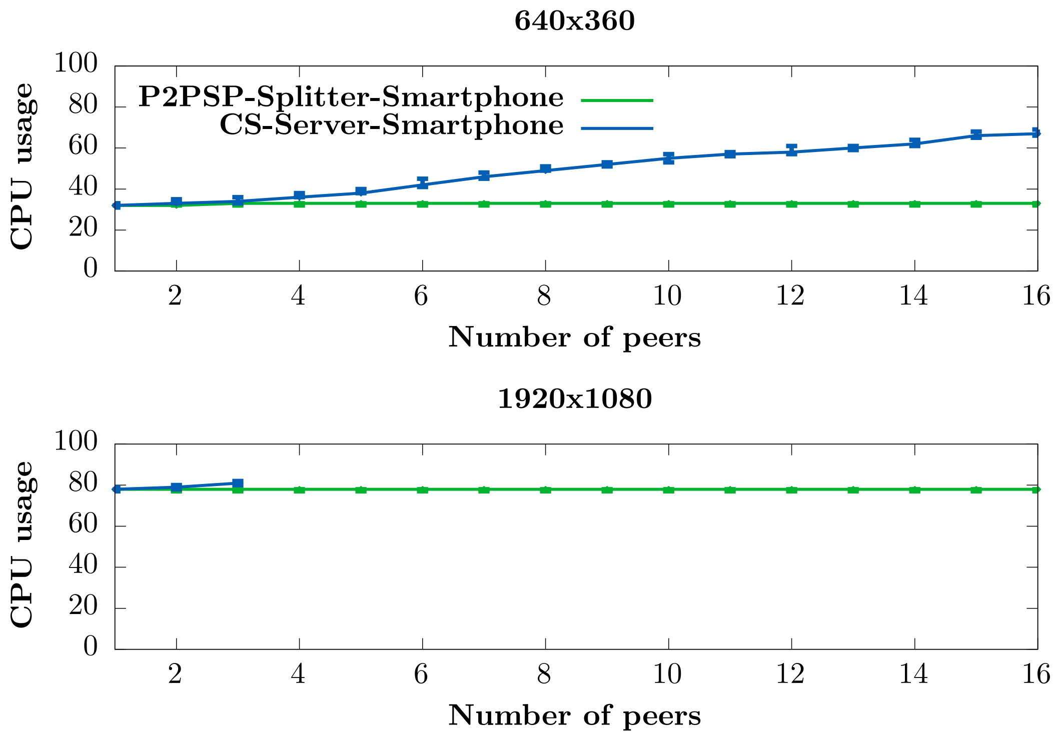


Figure 6.2: Comparison of CPU usage between a P2PSP splitter and a CS server running on a Moto X Play smartphone.

In the first experiment, we compare the CPU usage of the video server on a Moto X Play smartphone with Android 6.0.1 and Google Chrome v62.0.3202.84. The motivation of this experiment is to show the limitation of the CS model using a personal device without third-party video server. Both, the P2PSP splitter and CS server use WebRTC and data channel in a push mode. In this way, the Web browser is used to serve the video, avoiding the need of a Web server in the smartphone. Whereas the splitter just transmits one copy of the video, as described in Sec. 2 and Sec. 6.5, the server transmits several copies, one for each CS client. Figure 6.2 shows the CPU used by the splitter, and the server to deliver the video at each resolution. The splitter uses a constant CPU usage that depends on the video resolution but not on the number of peers. However, the CPU usage for the server increases with both, video resolution and number of clients. The server running in the smartphone is not able to deliver full HD video to more than three clients without loss of chunks. This is the main reason why third-party servers are used to deliver media stream.

The second experiment has been carried out to compare the computational resources required to run a P2PSP peer and a client in a GC. In order to be able to serve to the same number of peers and clients, the splitter and the server run now on a desktop PC. The peer and the client on the GC play the same media after requesting it via WebRTC and WebSocket, respectively. Both send chunks to the player in the same way (HTML video tag + MSE).

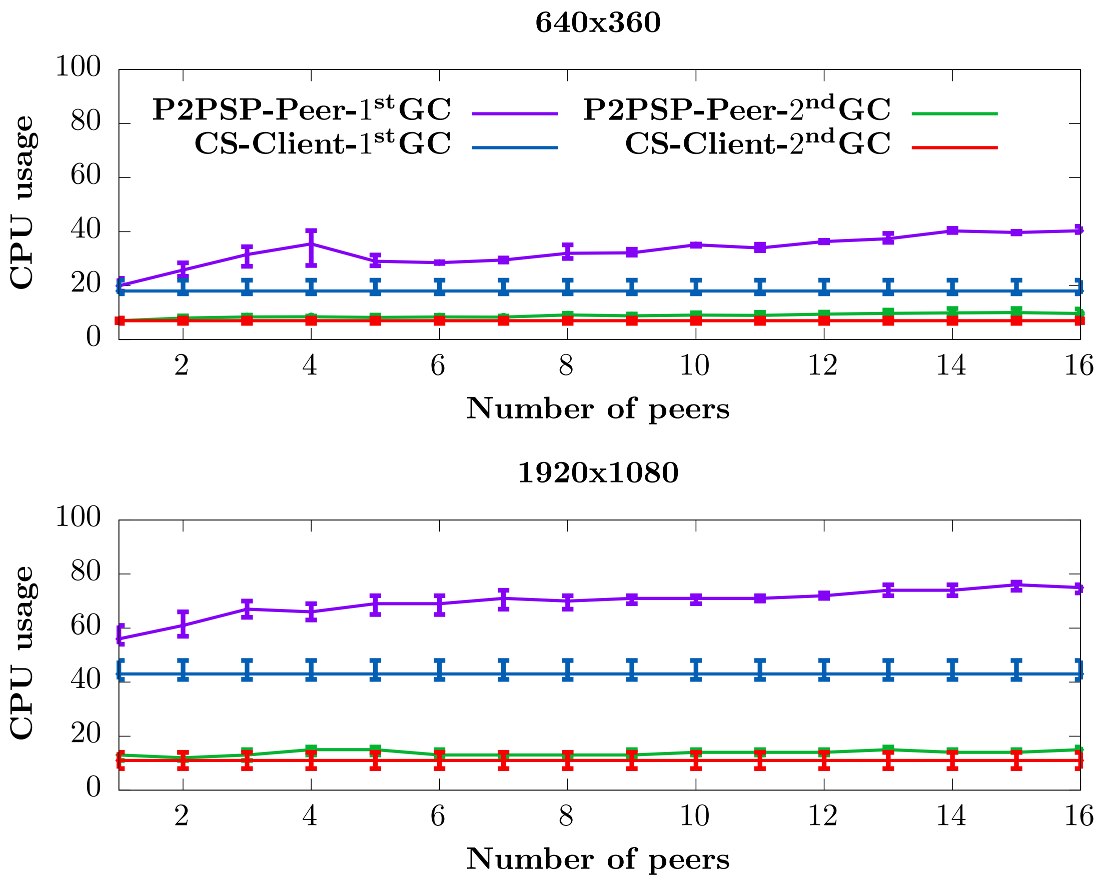
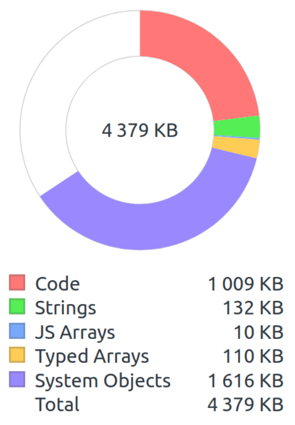


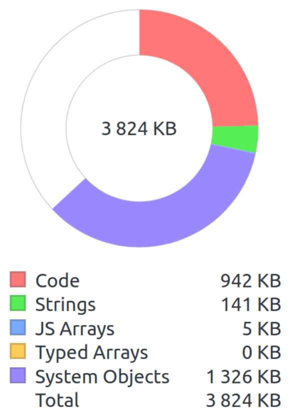
Figure 6.3: Comparison of CPU usage between a P2PSP peer and a CS client running on a 1st and 2nd generation of GC.

Figure 6.3 shows the CPU usage for a peer and a client running in the first two generations of GC. For each point plotted in the figure, we have performed five executions and the average, maximum and minimum of the CPU usage were taken from the GC developer console. Clients have a near constant CPU usage, depending on the video resolution and GC generation. Peers show a higher CPU usage than clients because, in addition to receive the video and send it to the player, they have to relay the chunks received from the splitter to the other peers in the team. Differences are very small in the second generation of GC.



🟥	Code	1 009 KB
🟩	Strings	132 KB
🟦	JS Arrays	10 KB
🟨	Typed Arrays	110 KB
🟪	System Objects	1 616 KB
	Total	4 379 KB

(a) P2PSP model.



🟥	Code	942 KB
🟩	Strings	141 KB
🟦	JS Arrays	5 KB
🟨	Typed Arrays	0 KB
🟪	System Objects	1 326 KB
	Total	3 824 KB

(b) CS model.

Figure 6.4: Memory usage of the WebRTC P2PSP peer vs WebSocket CS client.

Figure 6.4 shows the memory usage of a peer and a client. In both cases, the figures are similar. The only remarkable difference (apart from the System Objects, which shows the size used by other internals to the browser) is the space used by Typed Array. Clients store the ordered received chunks directly on the player buffer. However, peers need an additional buffer of Typed Array to sort the received chunks before moving them to the player buffer. This additional buffer depends on the team and chunk sizes.

6.7 Conclusions

Without a third-party server, the privacy of the streamed media is increased. However, due to the tendency of increasing the streaming media resolution, a server in the CS model running on a resource-constrained deviceis only able to serve a few number of clients. It is experimentally shown that a server using a P2P approach is able to run on resource-constrained devices. Additionally, the number of peers receiving the media increases significantly in comparison with a CS model. Moreover, a peer can run on a GC improving the QoE of the user when connected to an HDMI TV. The proposed P2P approach only uses web technologies and the only external server needed is the signaling one.

As future improvements, we propose to split the stream taking into account the media structure. Additionally, most important chunks should be delivered to more stable peers. In order to reach peers behind symmetric NAT devices, the use of a TURN server is currently mandatory. However, it does not follow the distributed communication scheme of P2P networks and new mechanisms without TURN server should be developed.

Chapter 7

Conclusions

This thesis dissertation has dealt with efficiency and security problems in P2P streaming protocols. In this chapter, we give a brief summary of the main issues and conclusions, and suggest some research directions that may be of interest for future work.

7.1 Efficency

At the beginning of this dissertation, we introduced P2PSP, an application-layer protocol that provides real-time broadcasting of a media stream on the Internet. Our proposal is performed taking into account previous studies carried out in the P2P streaming area. To maintain a straightforward structure and ensure an easy implementation, even in resource-constrained devices, we identified the main issues and proposed a modular design including load balancing, data broadcasting, IP Multicast, massively-lost chunk recovery, adaptative capacity, NAT traversal, multi channel, and content integrity set of rules.

In the efficiency aspect, the main concerns of this thesis are:

- NAT Traversal, a problem in P2P networks where peers need to know each other before establishing a communication. For treating it, we proposed a new and simple NAT Traversal algorithm that uses Collaborative Port Prediction (see Chapter 5). Experiment results showed that only a low number of guessed source ports is needed for traversal success. Also, two monitors peers collaborating leads to higher traversal success rates, compared with solutions with only two public servers.

- Running P2PSP on embedded devices (Chapter 6), which the aim of avoiding third-party media servers, in order to decrease the cost of the streaming and increase the degree of privacy. The proposed P2P approach only uses web technologies and the only external server needed is the signaling one. Our experiments showed that a server using a P2P approach is able to run on resource-constrained devices.

Our conclusion is that a P2PSP peer can be run in resource-constrained devices like Google Chromecast, and generally in any device able to run the web technologies in which implementation is based. Moreover, regarding NAT traversal, on the evidence of the success of our proposal, we are interested in applying it in real life NAT systems, as well as adapting it to more complex NAT addressing schemes.

7.2 Security

Regarding security, our efforts were focused on the content integrity set of rules, and particularly fighting against pollution attacks, a challenging security-related problem in peer-to-peer streaming platforms. Among the strategies proposed in this dissertation are:

- The use of trusted peers and digital signatures where attackers and defenders have to solve a multi-objective optimization problem and each decision made by one part may have a countermeasure by the other part. Our experimental results (Chapter 3) show that using only trusted peers as a defense strategy is not appealing when the number of malicious peers can be large.

- Enforcing peers to collaborate using Shamir's Secret Sharing (Chapter 3). After a theoretical analysis of the proposal, we conclude that the most severe possible attack is fully mitigated. For the remaining attacks, we can improve effectiveness to face them increasing the number of TPs.

- The use of a small number of trusted peers in Software Defined Networks by taking a proactive Moving Target Defense (Chapter 4). The experiments demonstrate the viability of our proposal, resulting in a relatively quick MP detection (in relation to the overlay size), not only in pure SDN environments but also in mixed environments where some peers are on the Internet and others are under managed networks.

Therefore, we can conclude that the best approach to face pollution attacks in fully-connected push-based peer-to-peer networks is the use of trusted peers under SDNs applying our studied malicious peer detection method. However, since modifying the scrambling period has a severe impact on the performance of the detection algorithm, we propose optimizations such as a proactive approach and controller location techniques as possible future lines of work.

Appendix

A

Publications arisen from this thesis

The research work carried out for the present thesis resulted in a number of publications. This appendix lists them along with their respective quality indicators and sorted by their year of publication (newest first) within each category

A.1 Publications in International Journals

[54] Medina-Lopez, C., Mertens, M.B., Gonzalez-Ruiz, V. & Casado, L.G. (2019). Reducing streaming cost while increasing privacy: A case study on a smartphone and chromecast using peer-to-peer technology to skip third-party servers. *IEEE Consumer Electronics Magazine*, DOI: 10.1109/MCE.2018.2880810. **8**, 50–55 Impact factor JCR 2018: 3.273. Journal ranking: 10/52 **(Q1) in *Computer science, Hardware & Architecture*.** 76/265 (Q2) in *Engineering, Electrical & Electronic*. 29/88 (Q2) in *Telecommunications*.

[-] *(Under Review)* An SDN Approach to Detect Targeted Attacks in P2P Fully Connected Overlays. International Journal of Information Security. Impact Factor JCR 2018: 1.822. **44/107 (Q2) in *Computer science, Software Engineering*. 42/104 (Q2) in *Computer science, Theory and Methods*.** 92/155 (Q3) *Computer science, Information systems*.

[-] *(Under Review)* NAT Traversal in P2P Networks using Collaborative Port Prediction. Peer-to-Peer Networking and Applications. Impact Factor JCR 2018: 2.397. 70/155 **(Q2) *Computer science, Information systems*.** 41/88 (Q2) *Telecommunications*.

A.2 Publications in Proceedings of International Conferences with DOI

[52] Medina-López, C., González-Ruiz, V. & Casado, L. (2017). On mitigating pollution and free-riding attacks by Shamir's Secret Sharing in fully connected P2P systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*, 711–716, DOI: 10.1109/IWCMC.2017.7986372. IEEE. GGS Rating: **B**, GGS Class: **3**.

[53] Medina-López, C., Shakirov, I., Casado, L. & González-Ruiz, V. (2017). On pollution attacks in fully connected P2P networks using trusted peers. In *Intelligent Systems Design and Applications*, 144–153, DOI: 10.1007/978-3-319-53480-0. Springer, Cham, Porto.CORE 2016: **rank C**.

[49] Medina-López, C., Casado, L. & González-Ruiz, V. (2015). Pollution Attacks Detection in the P2PSP Live Streaming System. In *International Joint Conference. CISIS 2015. Advances in Intelligent Systems and Computing*, 401–410, DOI: 10.1007/978-3-319-19713-5_34. Springer International Publishing. CORE 2014: **rank B**.

A.3 Publications in other International Conferences

[46] Medina-López, C. (2015). Participation as a speaker. In *GSoC 2015 Lightning Talks*, Google Inc., SunnyVale, CA. USA

[48] Medina-López, C., García Ortiz, J.P., Naranjo, J., Casado, L. & González-Ruiz, V. (2014). IPTV using P2PSP and HTML5+WebRTC. In *4th W3C Web and TV Workshop*, 5, IRT, W3C, Munchen, Germany

A.4 Publications in National Conferences

[51] Medina-López, C., González-Ruiz, V., Casado, L.G., Naranjo, J. & García-Ortiz, J.P. (2015). Ejecutando peers p2psp en google chromecast. In *Actas VI Jornadas de Computación Empotrada*, 123–129, Cordoba

[47] Medina-López, C., Naranjo, J., García-Ortiz, J.P., Casado, L.G. & González-Ruiz, V. (2013). Execution of the P2PSP protocol in parallel environments. In

G.B. y Alberto A. Del Barrio Garcia, ed., *Actas XXIV Jornadas de Paralelismo*, 216–221, Madrid

Appendix

B

Other publications produced during the elaboration of this Thesis

The research effor invested during the time span in which this thesis was elaborated produced some additional publications as the result of other research and education effors not included in this dissertation.

B.1 Publications in International Journals

[2] Andujar, A. & Medina-López, C. (2019). Exploring New Ways of eTandem and Telecollaboration Through the WebRTC Protocol: Students' Engagement and Perceptions. *International Journal of Emerging Technologies in Learning (iJET)*, **14**, 200–217

B.2 Publications in National Journals

[50] Medina-Lopez, C., Casado, L.G. & Gonzalez-Ruiz, V. (2015). P2PSP: un protocolo de red sencillo como herramienta para el aprendizaje basado en proyectos. *Experiencias Docentes en Redes de Computadores*, **1**, 35–41

Bibliography

[1] Albanese, M. & Huang, D. (2018). MTD 2018: 5th ACM Workshop on Moving Target Defense (MTD). In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 2175–2176, DOI: 10.1145/3243734.3243877. ACM.

[2] Andujar, A. & Medina-López, C. (2019). Exploring New Ways of eTandem and Telecollaboration Through the WebRTC Protocol: Students' Engagement and Perceptions. *International Journal of Emerging Technologies in Learning (iJET)*, **14**, 200–217.

[3] Antonatos, S., Akritidis, P., Markatos, E. & Anagnostakis, K. (2007). Defending against hitlist worms using network address space randomization. *Computer Networks*, DOI: 10.1016/j.comnet.2007.02.006. **51**, 3471 – 3490.

[4] Baccichet, P., Noh, J., Setton, E. & Girod, B. (2007). Content-aware p2p video streaming with low latency. In *Multimedia and Expo, 2007 IEEE International Conference on*, 400–403, IEEE.

[5] Bellovin, S.M. (1989). Security Problems in the TCP/IP Protocol Suite. *SIGCOMM Comput. Commun. Rev.*, DOI: 10.1145/378444.378449. **19**, 32–48.

[6] Bocek, T., Peric, D., Hecht, F., Hausheer, D. & Stiller, B. (2009). PeerVote: a decentralized voting mechanism for P2P collaboration systems. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, 56–69, Springer.

[7] Cai, G., Wang, B., Wang, X., Yuan, Y. & Li, S. (2016). An introduction to network address shuffling. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*, 185–190, IEEE.

[8] Carroll, T.E., Crouse, M., Fulp, E.W. & Berenhaut, K.S. (2014). Analysis of network address shuffling as a moving target defense. In *2014 IEEE International Conference on Communications (ICC)*, 701–706, IEEE.

[9] Chu, X., Zhao, K., Li, Z. & Mahanti, A. (2009). Auction-based on-demand p2p min-cost media streaming with network coding. *IEEE Transactions on Parallel and Distributed Systems*, **20**, 1816–1829.

[10] Cohen, B. (2001). `http://www.bittorrent.com`.

[11] Cohen, B. (2003). Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 68–72.

[12] Comer, D.E. (2000). *Internetworking with TCP/IP. Principles, Protocols, and Architectures (4th Edition)*, vol. 1. Prentice Hall.

[13] community, S. (2014). Swirl Project. `http://www.swirl-project.org`.

[14] Conner, W. & Nahrstedt, K. (2007). Securing peer-to-peer media streaming systems from selfish and malicious behavior. In *Proceedings of the 4th on Middleware doctoral symposium*, 13, ACM.

[15] Coughlin, T. (2017). How big are your dreams? gauging the size of future content [the art of storage]. *IEEE Consumer Electronics Magazine*, DOI: 10.1109/MCE.2016.2640620. **6**, 108–124.

[16] Dai, W. (2009). Speed Comparison of Popular Crypto Algorithms. `http://www.cryptopp.com/benchmarks.html`.

[17] Davis, M.H., Lang, U. & Shetye, S. (2015). A cybermodel for privacy by design: Building privacy protection into consumer electronics. *IEEE Consumer Electronics Magazine*, DOI: 10.1109/MCE.2014.2361192. **4**, 41–49.

[18] Deering, S.E. (1988). Host extensions for ip multicasting.

[19] Dhungel, P., Hei, X., Ross, K.W. & Saxena, N. (2007). The pollution attack in P2P live video streaming: measurement results and defenses. In *Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, 323–328, ACM.

[20] Diot, C., Levine, B.N., Lyles, B., Kassem, H. & Balensiefen, D. (2000). Deployment issues for the ip multicast service and architecture. *IEEE network*, **14**, 78–88.

[21] Edan, N.M., Al-Sherbaz, A. & Turner, S. (2017). Design and evaluation of browser-to-browser video conferencing in WebRTC. In *2017 Global Information Infrastructure and Networking Symposium (GIIS)*, DOI: 10.1109/GIIS.2017.8169813. 75–78.

[22] Eittenberger, P., Herbst, M. & Krieger, U. (2012). RapidStream: P2P streaming on android. In *Packet Video Workshop (PV), 2012 19th International*, 125–130.

[23] Gaeta, A., Kosta, S., Stefa, J. & Mei, A. (2013). StreamSmart: P2P video streaming for smartphones through the cloud. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*, 233–235.

[24] Gouda, M.G. (2014). Keynote talk: Communication without repudiation: The unanswered question. In *Networked Systems*, 1–8, Springer.

[25] Green, M., MacFarland, D.C., Smestad, D.R. & Shue, C.A. (2015). Characterizing network-based moving target defenses. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, 31–35, ACM.

[26] Han, J., Zhu, Y., Liu, Y., Cai, J. & Hu, L. (2005). Provide privacy for mobile p2p systems. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, 829–834, IEEE.

[27] Handley, M.J. & Rescorla, E. (2006). Internet denial-of-service considerations.

[28] Ho, C.Y., Wang, F.Y., Tseng, C.C. & Lin, Y.D. (2011). NAT-Compatibility Testbed: An Environment to Automatically Verify Direct Connection Rate. *Communications Letters, IEEE*, DOI: 10.1109/LCOMM.2010.102810.101700. **15**, 4–6.

[29] Hu, B. & Zhao, H. (2009). Pollution-resistant peer-to-peer live streaming using trust management. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, DOI: 10.1109/ICIP.2009.5414199. 3057–3060.

[30] IETF (2015). Peer to Peer Streaming Protocol (PPSP). `https://www.rfc-editor.org/rfc/rfc7574.txt`.

[31] Jafarian, J.H., Al-Shaer, E. & Duan, Q. (2012). Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, 127–132, ACM.

[32] Jennings, C. & Audet, F. (2007). Network Address Translation (NAT) behavioral requirements for unicast UDP (RFC 4787). *Network*.

[33] Kang, X. & Wu, Y. (2014). A trust-based pollution attack prevention scheme in peer-to-peer streaming networks. *Computer Networks*, DOI: http://dx.doi.org/10.1016/j.comnet.2014.07.012. **72**, 62–73.

[34] Killi, B.P.R. & Rao, S.V. (2017). Capacitated next controller placement in software defined networks. *IEEE Transactions on Network and Service Management*, DOI: 10.1109/TNSM.2017.2720699. **14**, 514–527.

[35] Kim, J. & Park, S. (2014). Resource Analysis for Mobile P2P Live Video Streaming. In *Ubiquitous Information Technologies and Applications*, vol. 280 of *Lecture Notes in Electrical Engineering*, 245–252, Springer.

[36] Lantz, B., Heller, B. & McKeown, N. (2010). A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 19:1–19:6, DOI: 10.1145/1868447.1868466. ACM, New York, NY, USA.

[37] Li, B., Xie, S., Qu, Y., Keung, G.Y., Lin, C., Liu, J. & Zhang, X. (2008). Inside the new coolstreaming: Principles, measurements and performance implications. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 1031–1039, IEEE.

[38] Lin, E., de Castro, D.M.N., Wang, M. & Aycock, J. (2010). SPoIM: A close look at pollution attacks in P2P live streaming. In *Quality of Service (IWQoS), 2010 18th International Workshop on*, DOI: 10.1109/IWQoS.2010.5542735. 1–9.

[39] Lin, W.S., Zhao, H.V. & Liu, K.R. (2010). Attack-resistant collaboration in wireless video streaming social networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, 1–4, IEEE.

[40] Lin, Y.H., Yang, C. & Chen, X.X. (2009). Design and implementation of Embedded P2P streaming media system. In *Machine Learning and Cybernetics, 2009 International Conference on*, vol. 3, 1364–1369.

[41] Lu, Z., Wang, Y. & Yang, Y.R. (2012). An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model. *Journal of communications*, **7**.

[42] MacDonald, D. & Lowekamp, B. (2010). NAT behavior discovery using session traversal utilities for NAT (STUN). Tech. rep.

[43] MacFarland, D.C. & Shue, C.A. (2015). The sdn shuffle: creating a moving-target defense using host-based software-defined networking. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, 37–41, ACM.

[44] Marti, S. & Garcia-Molina, H. (2003). Identity crisis: anonymity vs reputation in P2P systems. In *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, 134–141, IEEE.

[45] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. & Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, **38**, 69–74.

[46] Medina-López, C. (2015). Participation as a speaker. In *GSoC 2015 Lightning Talks*, Google Inc., SunnyVale, CA. USA.

[47] Medina-López, C., Naranjo, J., García-Ortiz, J.P., Casado, L.G. & González-Ruiz, V. (2013). Execution of the P2PSP protocol in parallel environments. In G.B. y Alberto A. Del Barrio Garcia, ed., *Actas XXIV Jornadas de Paralelismo*, 216–221, Madrid.

[48] Medina-López, C., García Ortiz, J.P., Naranjo, J., Casado, L. & González-Ruiz, V. (2014). IPTV using P2PSP and HTML5+WebRTC. In *4th W3C Web and TV Workshop*, 5, IRT, W3C, Munchen, Germany.

[49] Medina-López, C., Casado, L. & González-Ruiz, V. (2015). Pollution Attacks Detection in the P2PSP Live Streaming System. In *International Joint Conference. CISIS 2015. Advances in Intelligent Systems and Computing*, 401–410, DOI: 10.1007/978-3-319-19713-5_34. Springer International Publishing.

[50] Medina-Lopez, C., Casado, L.G. & Gonzalez-Ruiz, V. (2015). P2PSP: un protocolo de red sencillo como herramienta para el aprendizaje basado en proyectos. *Experiencias Docentes en Redes de Computadores*, **1**, 35–41.

[51] Medina-López, C., González-Ruiz, V., Casado, L.G., Naranjo, J. & García-Ortiz, J.P. (2015). Ejecutando peers p2psp en google chromecast. In *Actas VI Jornadas de Computación Empotrada*, 123–129, Cordoba.

[52] Medina-López, C., González-Ruiz, V. & Casado, L. (2017). On mitigating pollution and free-riding attacks by Shamir's Secret Sharing in fully connected P2P systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*, 711–716, DOI: 10.1109/IWCMC.2017.7986372. IEEE.

[53] Medina-López, C., Shakirov, I., Casado, L. & González-Ruiz, V. (2017). On pollution attacks in fully connected P2P networks using trusted peers. In *Intelligent Systems Design and Applications*, 144–153, DOI: 10.1007/978-3-319-53480-0. Springer, Cham, Porto.

[54] Medina-Lopez, C., Mertens, M.B., Gonzalez-Ruiz, V. & Casado, L.G. (2019). Reducing streaming cost while increasing privacy: A case study on a smartphone and chromecast using peer-to-peer technology to skip third-party servers. *IEEE Consumer Electronics Magazine*, DOI: 10.1109/MCE.2018.2880810. **8**, 50–55.

[55] Muller, A., Carle, G. & Klenk, A. (2008). Behavior and classification of NAT devices and implications for NAT traversal. *Network, IEEE*, DOI: 10.1109/MNET.2008.4626227. **22**, 14–19.

[56] Muller, A., Evans, N., Grothoff, C. & Kamkar, S. (2010). Autonomous NAT Traversal. In *International Conference on Peer-to-Peer Computing (P2P)*, 1–4, DOI: 10.1109/P2P.2010.5569996. IEEE.

[57] Ng, K.F., Ching, M.Y., Liu, Y., Cai, T., Li, L. & Chou, W. (2014). A p2p-mcu approach to multi-party video conference with webrtc. *International Journal of Future Computer and Communication*, **3**, 319.

[58] Nurminen, J., Meyn, A., Jalonen, E., Raivio, Y. & Garcia Marrero, R. (2013). P2P media streaming with HTML5 and WebRTC. In *Computer Communications Workshops*, 63–64.

[59] Petrocco, R., Pouwelse, J. & Epema, D.H. (2012). Performance analysis of the libswift p2p streaming protocol. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, 103–114, IEEE.

[60] Pfaff, B., Lantz, B., Heller, B., Barker, C., Beckmann, C., Cohn, D., Talayco, D., Erickson, D., McDysan, D., Ward, D. & et al. (2012). Openflow switch specification v1.3.1.

[61] Raju, H.S. & Salim, S. (2016). Reputation and trust management models in peer-to-peer. In *2016 International Conference on Research Advances in Integrated Navigation Systems (RAINS)*, DOI: 10.1109/RAINS.2016.7764398. 1–3.

[62] Rosenberg, J. (2010). Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols. https://tools.ietf.org/html/rfc5245.

[63] Rosenberg, J., Mahy, R., Matthews, P. & Wing, D. (2008). Session Traversal Utilities for NAT (STUN) (RFC 5389). https://tools.ietf.org/html/rfc5389.

[64] Ryu Project Team (2013). Ryu SDN Framework. `https://osrg.github.io/ryu/`, Accessed: 2018-07-10.

[65] Shabtay, L. & Rodrig, B. (2011). Ip multicast in vlan environment. US Patent 7,924,837.

[66] Shamir, A. (1979). How to share a secret. *Commun. ACM*, DOI: 10.1145/359168.359176. **22**, 612–613.

[67] Srisuresh, P. & Holdrege, M. (1999). IP network address translator (NAT) terminology and considerations (RFC 2663).

[68] Srisuresh, P., Systems, K. & Ford, B. (2008). State of peer-to-peer (p2p) communication across network address translators (nats). https://tools.ietf.org/html/rfc5128.

[69] Stutzbach, D. & Rejaie, R. (2006). Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, 189–202, DOI: 10.1145/1177080.1177105. ACM, New York, NY, USA.

[70] Takeda, Y. (2003). Symmetric NAT Traversal using STUN. https://tools.ietf.org/id/draft-takeda-symmetric-nat-traversal-00.txt.

[71] Team, T.P. (2014). Peer to Peer Straightforward Protocol Source Code. `https://github.com/P2PSP`.

[72] Team, T.P. (2016). P2PSP war-games repository. `https://github.com/P2PSP/war-games`.

[73] Vieira, A.B., Campos, S. & Almeida, J. (2009). Fighting attacks in P2P live streaming. simpler is better. In *INFOCOM Workshops 2009, IEEE*, 1–2, IEEE.

[74] Wang, C. & Lu, Z. (2018). Cyber deception: Overview and the road ahead. *IEEE Security Privacy*, DOI: 10.1109/MSP.2018.1870866. **16**, 80–85.

[75] Wei, Y., Yamada, D., S., Y. & Goto, S. (2008). A New Method for Symmetric NAT Traversal in UDP and TCP. In *APAN Network Research Workshop*, 11 – 18.

[76] Zhang, X., Liu, J., Li, B. & Yum, Y.S.P. (2005). Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, DOI: 10.1109/INFCOM.2005.1498486. **3**, 2102–2111 vol. 3.

[77] Zheng, J. & Namin, A.S. (2019). A survey on the moving target defense strategies: An architectural perspective. *Journal of Computer Science and Technology*, DOI: 10.1007/s11390-019-1906-z. **34**, 207–233.

[78] Zhou, X., Lu, Y., Wang, Y. & Yan, X. (2018). Overview on moving target network defense. In *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, DOI: 10.1109/ICIVC.2018.8492800. 821–827.