P2PSP (Peer-to-Peer "Straightforward" Protocol)

Cristobal Medina-López, J. A. M. Naranjo,
L. G. Casado and Vicente González-Ruiz
Universidad de Almería

June 8, 2015

The Peer-To-Peer Straightforward Protocol (P2PSP) is an application-layer protocol designed for real-time broadcasting of live media over a P2P overlay network. As many other P2P protocols, it minimizes the bandwidth requirements on the source nodes (which executes the process in charge of sending the data stream) by profiting of the upload bandwidth available in the links of the peers. However, the P2PSP has several characteristics that make it different of other previous proposals: (1) the protocol is simple enough to allow a straightforward implementation, (2) churn tipically produces a small number of lost blocks spreaded in the time and therefore, error concealment techniques based on signal interpolation can be applied more effectively, (3) peers can be behind NATs, (4) the protocol is media-agnostic, (5) the P2PSP is extremely modular which allows you to adapt it accurately to your requirements, (6) it can be deployed together client/server streaming services, among others advantages.

1 Introduction

Massive distribution of real-time video content is one of the big challenges in the Internet. Nowadays, there are several proposals that approximate to this goal, but none of them provide a QoS (Quality of Service) comparable to the DVB (Digital Video Broadcasting). This is a direct consequence of the design of the Internet and an unefficient use of the capacity of the network. The Internet was created to provide the so called best-effort service that basically 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, obviously, the amount of sent data. These two last factors are directly related to the capacity of the network, a term also refered 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 a linear growth of the transmmited data when the number of receivers increases. In this situation, P2P (Peer-to-Peer) overlays can improve the performance of the real-time streaming services: given that all peers manage (and thus can share) the same content, the trasmission requirements at the source are drammatically reduced.

This work introduces P2PSP (Peer-to-Peer Straightforward Protocol). P2PSP is a set of transmission and machine behavior rules that helps increasing 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 an 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.

2 Some networking facts

This section enumerates a list of facts that typically are found in those current data transmission scenarios which are based on computer networks.

1. **Routers deliever datagrams, not streams between connected devices:** In other words, the cost (in terms of bandwidth) of sending two or more packets to two or more different hosts is equal to the cost of sending two or more packets at the same host.

2. **Redundancy enables data compression:** Some data link protocols, such as the PPP (Point-to-Point Protocol), can compress the payloads in order to decrease the size of the packets. Therefore, assuming a constant packet size, it is expectable that the cost of sending two or more identical packets on content should be less than or equal to the cost of sending two or more packs of the same size but with different content.

3. **IP multicast availability:** Although IP multicasting is disabled at global scale, locally is usually available and is this case, it is the most efficient way of data broadcasting. For this reason, network level multicasting should be used whenever possible.

4. **Encapsulation overhead:** Each block of media content (which it will be referred as a "chunk" in the rest of this document) sent between peers must undergo a process of encapsulation which is basically be adding a header and sometimes a trailer for each network layer a packet traverses in his trip. The headers of the physical, data-link and network layers are compulsory in the Internet. However, at the transport layer level, there are basically two options: (1) the TCP (Transmission Control Protocol), a reliable protocol from the point of view of transmission errors and that avoids network congestion and (2) the UDP (User Datagram Protocol), which basically provides datagram transmission service. Apart from these differences, it should be noted that

the header overhead of both protocols is different: 20 bytes in the case of TCP and 8 bytes in the case of UDP.

5. **Congestion control and latency:** Internet is a shared medium and as such, the bandwidth provided is unknown a priori because it depends mainly of the bandwidth than other network users consume at that time. Furthermore, when the demand for bandwidth is higher than the network can provide, a phenomenon known as the network congestion occurs. This effect is a consequence of the routers, the devices that decide the paths to be followed by data packets, receive more data than can be process, and when this happens the packets are simply destroyed. This behavior brings a serious negative impact on overall network performance because, usually, a destroyed packet means that sooner or later it will be retransmitted and thus it will contribute to further congestion the network.

   To avoid getting into this dangerous dynamic, the TCP provides a mechanism for congestion avoidance which basically reduces the transmission rate if there are indications that the network is congested. As a result of the reduction of the transmission rate, users experience an increased latency in communication. In the case of UDP, such a mechanism does not exist and is the responsibility of the application to prevent the network congestion.

3 P2P architectures

The QoS (Quality of Service) provided by the streaming system depends strongly on the topology of the cluster of peers. Some proposals, such as Overcast [13] and Scattercast [5], define a two-tier network structure where some (supposedly high reliable) peers form a core network and the rest of peers are in an outer network. This architecture is, in essence, similar to a CDN because the outer peers do not contribute to the network and therefore, the scalability is quite limited. Additionally, if a core peer fails, several core and outer peers can experiment an interruption in the receiving of the stream.

Other approaches, such as PeerCast [7], Scribe [4], NICE [2], ZigZag [24] and BulkTree [1], distribute the peers in a tree (that in the end is a specific case of the star topology). It is well known that tree-push techniques are optimal respect to transmission delay, but only when the overlay structure matches the physical network structure. Besides, if a peer in the tree fails, then all the descendant peers will experiment an interruption in the streaming, effect that becomes larger when the failed peer is closer to the tree root. Finally, the leaf peers do not send data and therefore, the scalability of the system is limited. This last drawback can be mitigated building multiple trees and configuring leaf peers as root peers in other tree, approach that has been used in Zebra [8], CoopNet [19], SplitStream [3] and Orchard [17], albeit that the rest of problems remains.

In order to improve the robustness of the overlay, some protocols such as Narada [26], Yoid [11], PULSE [21] and Bullet [14] propose a tree/mesh-pull structure, similar to the one used in BitTorrent [6]. This architecture has been also

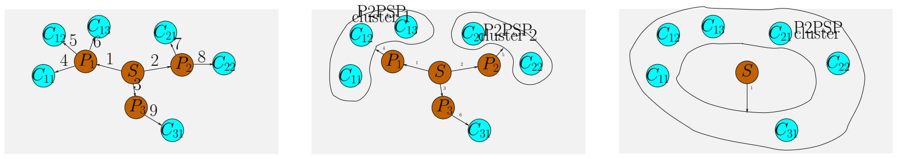

Figure 1: Different usage levels of the P2PSP. On the left, a pure C/S system where the content provider sends 9 copies of the stream to six clients C_{ij} by means of a server S and three proxies P_1, P_2 and P_3. On the righ, a pure P2PSP system where the content provider sends 1 copy of the stream. In the middle, a mix of both models which requires to send 6 copies of the stream.

utilized in the PPSP [12]. However, the signaling overhead that are necessary to maintain the dynamic structure of the mesh and the interchange of data between the peers (in order to perform the pull-operations) usually limites the size of the overlays, or at least, the neighbour-rate between the peers, a key aspect that usually helps to cope with the un-notified churn.

The proposal introduced in this document, the P2PSP, is an hybrid structure that can match any of the previously described topologies, depending on the user configuration. A pure P2PSP overlay lies in one of the extremes of this spectrum of P2P streaming algorithms because it can be clasiffied as a mesh-push structure with the maximal neighbour-rate because each peer send chunks of data to the rest of peers of the cluster (in other words, the diameter of the cluster is always 2). This provides to a P2PSP cluster a high error resilience, specially against churn. In the other extreme, a P2PSP overlay can consist of a large group of P2PSP teams which are interconnected with a star structure. A description of that idea has been depicted in Figure 1, where three different configurations have been shown. On the left one, there is a pure C/S structure. In the center, some proxies have reduced their bandwidth requirements thanks to the creation of some P2PSP clusters. In the right, the content privider only sends a copy of the stream by means of a pure P2PSP structure.

4 The Peer-To-Peer Straightforward Protocol

P2PSP [23, 16] is a application layer protocol for real-time streaming of multimedia content over the Internet, i.e., users playing the same stream in a synchronized way (all peers follow the same playback point). It can be used to build a variety of live-streaming services that ranges from small hangouts to large IPTV (Internet Protocol TV) systems.

4.1 Main P2PSP characteristics

These are some of the P2PSP features:

- P2PSP is not aware of the broadcasted content, the bit-rate, the format, etc. Any type of stream can be transmitted without having to modify the protocol at all.

- At least one working implementation of P2PSP can be found in Launchpad [22]. It can be used/modified/expanded without restrictions as long as the GNU GENERAL PUBLIC LICENSE [9] guidelines are followed.

- P2PSP has a modular architecture. The number of modules used depends on the requirements of the system to be deployed.

- The most basic module is simple enough to run the peer process in systems with very low computing resources. The rest of modules add functionality to the protocol, such as connectivity across NATs, parallel streaming, data integrity and information privacy.

- If native IP multicast is available (even locally, as it happens in most of the local area networks), the P2PSP can use it.

- The P2PSP facilitates the use of error concealment techniques in the received stream because lost packets are spreaded along the time.

- Peers can be hosted in private networks, even if they are placed behind symmetric NATs.

- The protocol is fully compatible with multiresolution and bandwidth-adaptive streaming services. Simulcast [**?**], scalable video coding [**?**] and multiple description video coding based solutions [**?**] are possible.

- P2PSP has been conceived for P2P real-time streaming services but it can be used to deploy hybrid C/S-P2PSP systems.

4.2 Data partitioning

In the P2PSP, data can be transmitted in two different states:

1. As a **stream**, i.e., as an endless sequence of data that transports some kind of information. Streams are always transmitted over TCP (Transmission Control Protocol).

2. As a **collection of chunks**, being a chunk a piece of stream. All chunks have the same size. Chunks are always transmitted over UDP (User Datagram Protocol). A small chunk minimizes the average latency of the transmission but also increments the underlying protocols (UDP/IP (Internet Protocol)/datalink) overhead, and vice versa.[1]

[1]Anyway, the chunk size should not exceed the MTU (Maximum Transfer Unit) of the transmission links in order to avoid the overload produced by IP fragmentation.

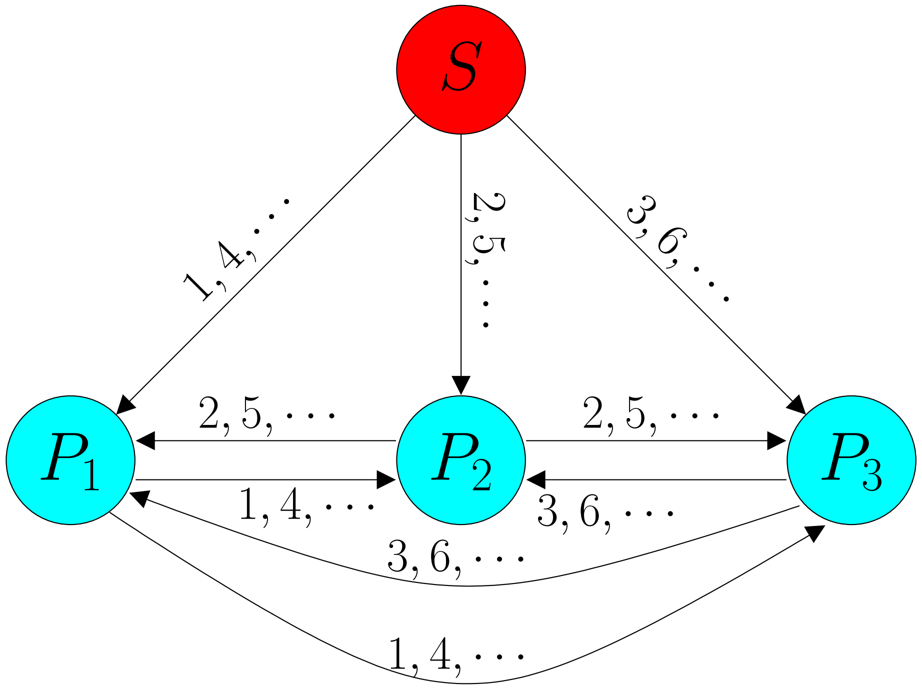

Figure 2: A P2PSP team. Arrows and their labels indicate the transmission of chunks. S sends one different chunk to one different peer which is selected using a Round-Robin model. Peers send each chunk received from S to each other peer in the team.

4.3 Basic entities

A P2PSP overlay network is composed the entities (in general, nodes) that are explained following:

1. **Source (O):** It is the producer of the stream which is transmitted over the P2PSP network. Typically, it is a streaming server using the HTTP protocol such as Icecast [10]. The source controls the transmission bit-rate in the P2PSP network and in general can serve to several clients.

2. **Player (L):** Players request and consume (decode and play) the stream. Usually, players can use signal interpolation techniques that are applied when some parts of the stream is missing. This is specially usefull in the P2PSP because lost chunks tend to be spreaded along time.

3. **Splitter (S):** This entity receives the stream from the source, splits it into chunks of the same size and sends the chunks to peers.

4. **Peer (P):** Receives chunks from the splitter and from other peers, ensembles the stream and sends it to a player. Some chunks are also sent to other peers (see below). Notice also that the player and the peer usually run on the same host.

5. **Team:** A splitter and one or more peers. As an example, Figure 2) show a P2PSP team of size 3 (the splitter is excluded of the computation of the team size).

4.4 IMS (IP Multicast Set of rules)

This set of rules implements the most basic behaviour of the protocol that can be used when IP multicast is available. This mode can be useful in local area networks where this transmission mode usually works properly.

1. **IP multicast address:** The splitter uses an IP multicast address (and a port) where all peers of the team wait for receiving chunks. Notice that the peers only receive chunks (never send them to the multicast channel because the splitter does all the work).

2. **Peer arrival:** An incoming peer must contact with the splitter in order to join the team. After that, the splitter sends to the peer the stream header over the TCP using a temporal unicast connection.

3. **Buffering in peers:** Packets in transit can suffer different transmission delays due to the jitter[2], even producing that they arrive out of order. For this reason, the splitter enumerates every chunk of stream with a 16-bit counter, producing a packet with the format:

   IMS_packet = [chunk_number, chunk]

[2]Variations in the network latency.

Peers store the received IMS_packets in a buffer whose size b can be different in each peer depending on the maximun tolerated delay.

4. **Relation between the buffer size B and chunk number upper bound M:** Due to practical reasons, the upper bound for the chunk number should be a power of two. In order to minimize the probability of receiving two or more chunks with the same number (remember that chunks can be reordered in transit), M must be a multiple of B, i.e.:

$$M = pB, \tag{1}$$

where $p \in \mathbb{N}$.

4.5 DBS (Data Broadcasting Set of rules)

This set of rules 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.

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

$$(i + n) \bmod |T| = 0, \tag{2}$$

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 retransmitted.

2. **Congestion avoidance in peers:** Each peer sends chunks using a constant bit-rate strategy to minimize the congestion of its uploading link. Notice that the rate of chunks that arrive to 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 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 is used by the peers to forward the received chunks to the other peers.

5. **Peer arrivals:** An incoming peer X must contact with the splitter in order to join the team. After that, the splitter sends to X the list of peers and the current[3] stream header over the TCP. More exactely, the splitter does:

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

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

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

   (c) Append X to the list of peers.

   In incomming peer X performs:

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

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

      i. Receive (TCP) end endpoint P_i fron the splitter.

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

   Because the [hello] messages can be lost, some peer of the team could not know X in this presentation. However, because peers also learh about their neighbors when a [IMS] message is received, the impact of these lost 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 (infra-solidary) peer can not enforce 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 pf 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 a impolite churn (those peers that leave the team without sending the [goodbye] message).

7. **Monitor peers:** Some peers (see P_0 in Figure 3), which usually run close (in hops) the splitter, play different roles depending on the P2PSP modules implemented. Among others:

   (a) As a consequence of the impolite churn and peer insolidarity, 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 adminitrator should monitorize the streaming session because, 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.

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

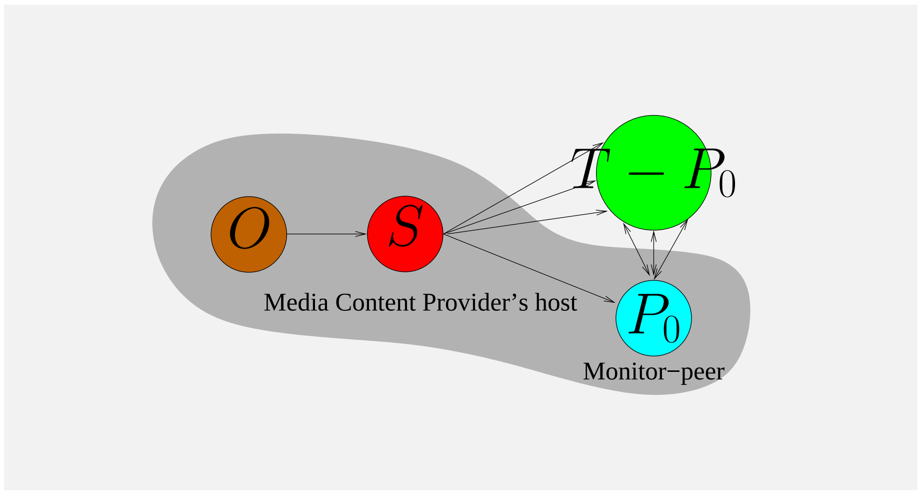

Figure 3: A typical P2PSP configuration using a monitor-peer (P_0). Notice that the monitor-peer, the source and the splitter run on the same host.

(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 of all peers (included the monitor peers, obviously) of the team is:

$$B\frac{|T|}{|T|+1}. \tag{3}$$

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

$$\lim_{|T|\rightarrow\infty} B\frac{|T|}{|T|+1} = B, \tag{4}$$

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

(c) In order to minimize the number of loss reports (see Rule 10) Section 4.7) 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

the splitter can 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 succession of events:

(a) In the rest of peers $\{P_j, i \neq j\}$, the free-riding control mechanism (see Rule **??**) will remove P_i from the list of peers.

(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 complains, the splitter will delete P_i from his list.

9. **Relation between the buffer size B and the team size $|T|$:** As in the IMS module, 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 [IMS] messages 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 X in the list of peers of the peer Y, it can last more or less chunk times for Y sending the [IMS] message to X.

   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

$$|T| \leq B. \tag{5}$$

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 lost chunks x to the splitter using [lost chunk number x] complain report messages. In the DBS module, a chunk is clasiffied 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 4). In order to guarantee this, the splitter counts the number of complains (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 6.

4.6 ACS (Adaptive Chunk-rate Set of rules)

All nodes of a team (peers and splitter) that implements only the DBS transmitt exactly the same amount of media, which basically implies that if a peer can not fulfill this requirement it will be thrown of the team. Unfortunately, this could be considered too much demanding in some specific configurations, such as a team of colleagues that want to share a content regardless of who spends more transmission bandwidth, or in PPV (Pay-Per-View) systems where the stream must be guaranteed to those users that have paid for receiving the stream.

The number of chunks that ultimately a peer must retransmit depends exclusively on the number of chunks that the peer receives from the splitter. Besides, the splitter knows the performance of the peers of the team for the task of retransmitting the chunks by checking the number of times that the peers has lost a chunk (see Section 4.7). This knowledgment could be exploited by the splitter to maximize the profiting of the team capacity by assigning a different splitter-chunk-rate[4] to the peers depending on their reliability. In other words, if a peer does not loss chunks, then its splitter-chunk-rate will be increased and viceversa. Now, lets classify the peers into two types: (1) class-A peers that contribute more and (2) class-B peers that contribute less.

By default and using only the ACS, the splitter-chunk-rate per peer will be the same for all peers if the team size remains constant. In this framework, the ACS proposes an adaptive Round-Robing scheduler (at the splitter, see Rule 1) in which the team cicle of a peer P is proportional to the packet loss ratio of P. Using the ACS, a class-A peer will receive from the splitter more chunks than a class-B peer and therefore, a class-A peer will enter in the burst mode (see Rule 3) more often than a class-B peer. This is something that goes against the throughtput of the class-A peer and that, in some moment, could produce a lost of chunks (remember the the burst mode can congest the upload link of the peers). Therefore, the throughtput of a class-A peer will grow until reachint its congestion threshold, instant in which the monitors peers will report the lost of this class-A peer and the splitter will decrease its splitter-chunk-rate.

Another consecuence of implementing the ACS is that class-A peers will remove from their list of peers to class-B peers more often, with a frequency that depends on among other things of the `MAX_LOSS_COUNTER`[5] configuration parameter of each peer. However, this action has not a noticeable impact on the performance of the team because the time that lasts from a class-A peer removes a class-B peer of its list of peers is smaller than the time that the class-B peer needs to send a chunk to the class-A peer, and when this happens, the class-B peer is inserted again in the list of peers of the class-A peer, reseting its loss counter. Anyway, an increment of the `MAX_LOSS_COUNTER` in class-A peers would improve the performance of the system.

[4]The number of chunks per second that the splitter sends to a peer.

[5]Each peer has a counter for each other peer of the team. This counter is increased when a chunk is sent to the peer and decreased when a chunk is received from that peer. If this counter is higher than `MAX_LOSS_COUNTER`, the unsupportive peer is removed from the list of peers (move this to DBS).

4.7 LRS (Lost chunks Recovery Set of rules)

The P2PSP relies on the UDP as the transport protocol and, obviously, packet losses can happen. The impact of a packet loss in the QoS offered by the team depends on where the packet is lost. If the packet is lost in its trip between the splitter and a peer, this packet will be missed by all the peers of the team, included the monitor peers. However, if the packet is loss in the trip between two peers, only the destination peer will loss the chunk.

Only monitor peers tell the splitter which chunks have not been received on time. More specifically, a monitor peer P sends to the splitter a [lost chunk index x] loss report message when a chunk with chunk-number x has been lost. Using this information, the splitter can enumerate the number of times that a chunk has been lost. In this framework, the LRS module defines that the splitter resend a lost chunk stored in the location $x \bmod M$ of its buffer of chunks if the number of losses is equal to the number of monitor peers. The selected peer to resend this block will one of the monitor peers.

Notice also that, in this case, monitor peers should become aware of a chunk loss some time before of that the rest of peers of the team send it to their players, in order to have enough time to resend the lost chunk from the monitor peer to the rest of peers of the team. A simple technique that has been proven to work is to use in the monitor peers a buffer size of half the size of the buffer size of the rest of peers. Thus, when a monitor peer realizes that a chunk has been lost, the rest of peers are receiving those chunks that are approximately in the middle of their buffers. Now, if the buffer if large enought, the resent chunks should be received on time. This implies also that, if the LRS module is implemented, the buffer sizes in (standard) peers should be doubled and therefore, it holds that

$$B \leq 2|T|. \tag{6}$$

4.8 EMS (End-point Masquerading Set of rules)

It is expected that most of the peers are running behind NAT (Network Address Translation) devices which connect private address networks to the public Internet. When a packet crosses the NAT from a private network towards the public one, the source (private) end-point is replaced (masqueraded) by a public end-point of the NAT and an translation entry is created in the NAT table. NAT translation entries are needed to relate the NAT public end-point with the source (private) end-point.

Basically, there are three types of NATs: (1) full-cone NATs, (2) restricted-cone NATs and (3) symmetric NATs. Depending on the type of the NAT, the number of fields in the NAT entry and the NAT behaviour is different. A Full-Cone NAT Entry (FCNE) has three fields:

$$\text{FCNE} = (\text{public NAT port } \mathcal{X}, \text{ private IP address } \mathcal{Y}, \text{ private port } \mathcal{Z})$$

and whatever the origin of the incoming packet[6], if that packet is received by the NAT at the end-point (public NAT IP address, public NAT port $\mathcal{X}$), the packet

[6]Incomming packets go from the Internet towards the private network.

will cross the NAT and it will be delivered to the proccess that is listening at the end-point (private IP address $\mathcal{Y}$, private port $\mathcal{Z}$).

This procedure is fully compatible with the DBS module because a full-cone NAT-ed peer[7] behaves like a public peer except that the NAT masks its actual private IP address. Nevertheless, a problem arises when two or more peers are behind the same NAT (are in the same private network). Although an efficient (but also complex) solution for this case is proposed in Section7.6, it could happen that this solution can not be applied. Therefore, if two (or more) peers A and B are in the same NAT-ed network and no NAT loopback[8] is avaiable, the DBS module does not provide enough functionality because the neigbouring peers does not know the private end-point of each other: A only knowns the public-end point of B and viceversa.

For providing the extra functionality to solve this situation peers must implement the EMS. At the beginning of the joining stage, each EMS-powered peer sends to the splitter its local end-point and the splitter checks if the source end-point of the received packet (which figures in the packet header) matches the local end-point. If these values are the same then the peer is public; otherwise, the peer is running in a private host. When this is true, it holds that

$$X \neq (X), \tag{7}$$

where X denotes the local (private) end-point of peer X and (X) the global (public) end-point of peer X in X's NAT. In general, we have also that

$$\mathcal{N}(T) \subset T, \tag{8}$$

where T represents all the elements of a team (including the splitter) and $\mathcal{N}(T)$ those peers that behind a NAT. In other words,

$$\mathcal{N}(T) = \{P \in T \mid P \neq (P)\}. \tag{9}$$

Notice also that the splitter can find out if a peer A has a neightbour B because in this case, the public IP address of the end-point that the splitter see of A and B matches.

Accordingly, when the splitter is sending the list of peers to a EMS-graded peer A and this peer is hosted by a private machine, the splitter also checks whether A has neighbours, and if this is true, the splitter sends to A the private end-point of B instead of its public end-point, and viceversa. Hence, A will use the private end-point of B to communicate with it and viceversa.

4.9 NTS (NAT Traversal Set of rules)

Connection-filtering-NATs are becoming increasingly frequent, and this is a situation that dificults the connectivity between peers. This set of rules introduces the

[7]A peer that is behind a full-cone NAT.

[8]This feature allows a peer A to connect to other peer B in the same NAT-ed network using a the public end-point of B in the NAT.

extra functionality to handle those peers that are behind restricted-cone NATs and symmetric NATs.

The DBS module enables the communication for those peers that are behind full-cone NATs and peers that are behind those more restrictive NATs but that have dedicated an open port for the P2PSP traffic, but in the rest of the cases NATs will block the incoming packets. In order to known the reason of that problem, let's examine the behaviour of restricted-cone and symmetric NATs.

When a restricted-cone NAT is used, the NAT entries can have four or five fields, depending on the exact type of NAT. Compared to a full-cone NAT, restricted-cone NATs entries have a fourth field which memorizes the destination IP address of the outcoming packet[9], i.e., we have a translation entry such as:

RCNE = (public source IP address $\mathcal{W}$, FCNE)

where RCNE stands for Restricted-Cone Nat Entry. Therefore, an incoming packet can cross the NAT only if it comes from a proccess that is running at a host whose IP address is $\mathcal{W}$.

If the NAT is a port-restricted-cone one, NAT entries become:

PRCNE = (public source port $\mathcal{V}$, RCNE)

where PRCNE stands for Port-Restricted Cone Nat Entry. In this case the NAT forwards the packet only if it was originated at the (public source port $\mathcal{V}$, public source IP address $\mathcal{W}$) end-point.

It is important to notice that in a cone NAT (restricted or not) only a public NAT port is assigned to each (private IP address $\mathcal{Y}$, private port $\mathcal{Z}$) end-point. This means that, although a peer is behind a cone NAT, the peers can be addressed using the unique public NAT end-point that the NAT has selected to create the corresponding translation entry. Thus, in order to reach the NAT-ed peer the only action that must be previously performed is to send a packet from this (private) peer to the interlocutor (public) peer, something that is already performed in the DBS.

Let us suppose that a new peer X wants to join the team. Following the DBS, when X is joining the team it receives the list of peers and sends a [hello] message to each peer of the list. These messages will be received by those peers that run at public hosts or full-cone NAT-ed hosts, but not otherwise.[10] To solve this problem, when a peer X want to join the team, the splitter can send to the all NAT-ed peers of the team[11] the X's end-point using the message:

[say hello to (X)],

where (X) is the public X's end-point asigned by its NAT. Thus, when the peers receive this message will send a [hello] to (X), creating a translation entry in their

[9]Outcoming packets go from the private network towards the Internet.

[10]Notice also that, if X is behind a NAT, these [hello] messages creates one or more translation entries in the NAT of X that makes possible that the rest of peers of the team reach X.

[11]Notice that in order to apply this rule, the splitter must know if a peer is behind a NAT or not. For this reason, it is compulsory that a NTS-graded peer implements also the EMS (see Section 4.8).

NATs. After that, X will be able to communicate with all the peers of the team, even if X is behind a NAT.

Summarizing, when an arriving peer X wants to join the team T and after receiving the list of peers in T from S, S must carry out the steps refered in Algorithm 1 and each peer in $\mathcal{N}(T)$ must follow the steps in Algorithm 2.

Algorithm 1 : NTS algorithm for S.

```
1. For each peer $P$ in $\mathcal{N}(T)$:
    (a) Send [say hello to $(X)$] to $P$.
```

Algorithm 2 : NTS algorithm for each peer P in $\mathcal{N}(T)$.

```
1. Receive [say hello to $(X)$] from $S$.
2. Send [hello] to $(X)$.
```

Unfortunately, symmetric NATs behaviour is different and the previously proposed algorithm does not work. A symmetric NAT assigns a different public port for each (public source port, public source IP address, private IP address, private port) combination. This means that, if a peer that is behind a symmetric NAT it will use different public NAT ports for communicating with any other peer and the splitter. Moreover, the algorithm used by a symmetric NAT to allocate the public ports is not standardized. Some NATs will assign ports sequentially (depending on their availability) and other will assign them at random. Notice also that symmetric NATs, by definition, are incompatible with the pure P2PSP philosophy. Each peer must sends messages to the rest of peers of the team and this means that $|T|$ ports will be allocated for each peer that is behind a symmetric NAT. For all these reasons, in order to run the P2PSP under symmetric NATs, the configuration presented in Section 7.6 should be used. However, if this solution if not factible, $|T|$ is small and the number of peers behind the symmetric NAT is also small, the following simple solution could be tried.

The first problem to solve here is to identify those peers that are behind symmetric NATs because their NAT traversal set of rules will be different of the rules that must follow those peers that are behind cone NATs. To indentify a symmetric NAT-ed peer X, the splitter sends to the monitor peer the public end-point of X and the monitor peer search this end-point in his list of peers. If the peer is in the list, X is behind a cone NAT, otherwise, X is behind a symmetric NAT. Algorithm **??** sumarizes this procedure.

Algorithm 3 : NTS algorithm for S to determine the type of NAT that an incomming peer X uses.

```
1. Send [$(X)$] to $P_0$.
2. Receive [type of $X$'s NAT].
```

Most NATs use the *port preservation* port allocation technique which means that if a process that runs in the (NAT-ed) private network uses a local port $\mathcal{L}$,

Algorithm 4 : NTS algorithm for P_0 to determine the type of NAT that an incomming peer X uses.

```
1. Receive[$(X)$] from $S$.
2. if $(X)$ is in $T$, then:
    (a) Send [$X$'s NAT is a cone NAT] to $S$.
3. else:
    (a) Send [$X$'s NAT is a symmetric NAT] to $S$.
```

then the NAT will try to use the same port at the public side, and if the public port $\mathcal{L}$ has been already allocated then the NAT will check if port $\mathcal{L}+\infty$ is free. Supposing this and taking into account that using the rules 5 and **??** of the DBS will allocate the ports $\mathcal{L}, \mathcal{L}+1, \cdots, \mathcal{L}+|T|$ where, $\mathcal{L}$ will be the port assigned for the NAT to talk with the splitter, $\mathcal{L}+1$ will be the port assigned to talk with the first peer of the list of peers and $\mathcal{L}+|T|$ will be the port to talk with the last peer of the list of peers. Notice that it is possible to determine whether the ports used by the symmetric NAT of the incoming peer X are $\mathcal{L}, \mathcal{L}+1, \cdots, \mathcal{L}+|T|$ after receiving the [hello] messages from X to the rest of peers of the team, even if there are peers behind symmetric NATs, provided these peers have sent a [hello] towards X using the corresponding public port at X's NAT. Therefore, if after this phase of "regards" between X and the rest of peers of the team, all these peers have received a [hello] from X, then we can conclude that X has joined to the team correctly.

Algorithms 5 and 6 sumerize the steps that must be performed in order to incorporate to a team a peer X that is behind a symmetric NAT.

Algorithm 5 : NTS algorithm for S when there are peers behing symmetric NATs.

```
1. Sort the list of peers.
2. For each peer $P$ in $\mathcal{S}(T)$:
    (a) Send [say hello to $(X) + \#P$] to $P$.
```

Algorithm 6 : NTS algorithm for each peer P in $\mathcal{S}(T)$ when P is behing symmetric NAT.

```
1. Sort the list of peers.
2. Receive [say hello to $(X) + \#P$] from $S$.
3. Send [hello] to $(X)$.
```

Finally, we would like to stress that this algorithm must be executed only by those peers that are behind a (port-)restricted-cone or symmetric NAT. The rest of peers of the team do not need to be aware of the use of these techniques.

4.10 MCS (Multi-Channel Set of rules)

The P2PSP may broadcast different channels (streams) over distinct (unconnected) teams and users can run several peers in parallel, one peer per channel the user wants to receive. The only essential requirement to enable multi-channel capability is the network providing sufficient bandwidth. However, when this condition can not be achieved, users should define the channel priorities. Using this information and if not enought bandwidth is avaiable, the peer instances that correspond to the lower priority channels will be identified as unsupportive peers and rejected from their teams.

To implement that behaviour, the MCS module introduces the encapsulation scheme:

[MCS] = [Priority, [DBS]]

and a new type of node, the Multichannel Scheduler M.

Therefore, those peers that implement the MCS must label each DBS message. This label is a 16-bit positive integer number that represents a priority, being zero the highest one. MCS messages are sent to M which basically implements a priority FIFO queue of messages. Each time a new [MCS] is received by M, it sorts them by priority and next, by chunk number. Thus, if there is not enough bandwidth to transmit all packets, the user stops receiving those channels that have been assigned a lower priority.

The MCS module can also be useful when simulcasting, scalable media coding and multiple description media coding is used. In these situations, the different channels refeer to different representations (qualities, resolutions, etc.) of the same media content.

4.11 The Content Integrity Set of rules

The CIS (Content Integrity Set of rules) is responsible for fighting against a DoS (Denial of Service) attacks by stream spoiling (also known as pollution attacks). This action could be carried out by possible custom implementations of peers that might to poison[12] (by altering willfully) the content of the stream. This set of rules could be also useful in those situations where the transmission links are error-prone and the error detection mechanism of the underlaying transport protocol has been disabled.

In the CIS is proposed use a hash of the content of Chunks to discover a attacker peer. The rules are:

1. One or more peers of the team are selected as trusted peers so that only the splitter knows of its existence through of endpoint of each them. It's possible that all peers in the team are trusted-peers except the attacker.

[12]A poisoned chunk is a chunk that seems to be OK, but which the sender has changed in such a way that when played, introduces no information (for example, a chunk filled with zeroes) or even wrong information.

2. The trusted peers create a hash (fingerprint) for a number of received chunks (included the chunk number) plus an other hash of the endpoint from where each chunk has been received. Depending on the computational power available in the trusted peer host, all or a subset (can be random) of chunks are processed.

3. The hashes (both chunks and endpoints) are sent to the splitter, which checks if the received chunks have been altered (calculate the hash is necessary).

4. The splitter knows what chunk has been sent to each peer. Therefore if the splitter receives a hash that does not match the one he has calculated can deduce that one of the chunks was altered and depending on the number of corresponding chunk is able to determine to which peer was sent the altered chunk (note that all chunks follow the following process: the chunk first travels from the splitter to a peer which sends it to all other peers of the team).

5. When the number of altered/peer exceeds a treshold, the peer is rejected of the team. This is achieved not sending more chunks to the attacker(s) peer(s). Moreover the splitter sends a reject message that contain the endpoint of the attacker to all peers of the team, this ensures that the attacker is removed from the peers list of all peers of the team as soon as possible.

4.11.1 A model of the impact of an attack

This mathematical model estimates the averages of poisoned chunks X into a team depending of number of trusted peers T, the numer of attackers peers A concurrently in a team and the P number of total peers (attackers or not) in the team. In addition, the model estimates the number of poisoned chunks that arrives to any peer, always in average values.

As noted in the begin of this section, the identity of the trusted peers is unknow for all except for the splitter. Moreover, the behavior of the attackers will be poison the maximun number of chunks. Note, however, that any intermediate selective situation with the chunks poisoned can be consider similar to this one (are poisoned all possible chunks) where the attackers number is lower.

Suppose initially that $T = 1$ (only exist one trusted peer in the team). In the more favorable situation (and unlikely) for an attacker, this could reach up to $P - 1$ chunks if in the retransmission cycle the last chunk is sent to the only one trusted peer. Moreover, It may also happen that the first poisoned chunk sent by an attacker arrives to an only one trusted peer. In this case, only one chunk is poisoned. As the position of the peers is random, the average number of poisoned chunks when $A = 1$ and $T = 1$ is

$$X = \frac{P - 1 + 1}{2} = \frac{P}{2} \tag{10}$$

Suppose that exist more of one trusted peer ($T > 1$ and $A = 1$). As now the probability of deliver a poisoned chunk to a trusted peer increment proportionality with T, the average number of poisoned chunks would be T times lower, i.e., the average number of poisoned chunks would be

$$X = \frac{P}{2T} \tag{11}$$

Finally, if there is more of one attacker ($T > 1$ and $A > 1$), that amount would be multiplied by A (suppose that the A attackers poisons the chunks in parallel), getting

$$X = \frac{AP}{2T} \tag{12}$$

From this expression can be derived two hypotheses. The first one, that the impact of an attack depends of the ratio between number of attackers and trusted peers (expected behavior). And second, that when A and T are of the same order the average poisoned chunks tend to be $P/2$ In the case of exist also normal peers, clearly X will increase. For example, if there is a friendly peer too, X will increase in a poisoned chunk per each concurrently attacker in the team. Therefore, it's determined that

$$X = \frac{AP}{2T} + (P - A - T) \tag{13}$$

As seen, the latter term does not significantly affect the average number of poisoned chunks, unless the team is very large, in which case, the attack is diluted because never the number of received chunks for each peer in the same retransmission cycle can be bigger than A.

4.12 The Data Privacy Set of rules

The following set of rules deals with privacy-related issues. Many content providers offer pay-per-view channels as part of their services. From a technical point of view, this implies having a Key Server that ciphers the stream with a symmetric encryption key and delivers such key to authorized members only. However, this is not enough: it is crucial that the Key Server renews the encryption key after the expiration of a peer's authorization period so the stream can not be decrypted any more by the peer (this feature is called *forward secrecy*). In addition, if we want to play on the safe side then the Key Server should renew the encryption key after a peer purchases an authorization period (if the key remained the same then the peer might decrypt previously captured stream packets for a later viewing). This renewal process is not trivial and is carried out by a *secure multicast protocol*. In order to alleviate the overhead incurred by avalanches of peers entering and leaving the authorized group (for example, at the beginning of a high interest event such as The Olympics) key renewal can be performed on a batch manner, i.e. renewing the key at a given fixed frequency rather than on a per arrival/exit basis. Finally, key renewal messages should be authenticated by means of a digital signature or other alternative methods [20].

Many secure multicast protocols protocols exist in the literature, for example [25, 15, 28, 27]. Here we suggest the implementation of a protocol by Naranjo et al [18]. On it, every authorized peer receives a large prime number from the Key Server at the beginning of its authorization period (this communication is done under a secure channel, for example SSL/TLS). For every renewal, the Key Server generates a message containing the new key to be used by means of algebraic operations: all the authorized primes are involved in this message generation process, and the key can only be extracted from the message by a peer with a valid prime. This protocol is efficient and suits P2PSP architecture in a natural way: every splitter can act as a Key Server for its own team. Hence, the stream would be first transmitted among splitters (possible encrypted by a different key, shared by the splitters). Within each team, its corresponding splitter would control the encryption and key renewal process.

4.13 The Peer-list Compression Set of rules

S maintains the list of peers $\{T\}$ ordered according to the numerical order of the IP addresses of the end-points of the peers. Thanks to this and depending on the compression ratio, the splitter should minimize the transmission time of the list of peers by compressing it with the Algorithm 7. Notice that only the IP addresses of the end-points are compressed.

Algorithm 7 : Compression algorithm of the list of peers.

```
1. Send 4 bytes for $T(0)$.
2. Send 2 bytes for "A", "C" or "C", depending on type of network that is going
   to be sent next.
3. Send 3 (A class), 2 (B class) or 1 (C class) bytes to encode the rest of peers
   of $T$ that are in the sub-network specified in the previous step.
4. For each peer in the sub-network:
   (a) Send the host part of the IP address of the peer, using 3 (A class), 2 (B
       class) or (C class) 1 bytes.
   (b) Send the port of the peer using 2 bytes.
```

5 A mathematical analysis of the P2PSP

Important aspects to analyze are: (1) the average throughtput of the system, measured as a ratio between the capacity of the network and the effective bandwidth that the overlay provides, (2) the average start-up/swiching delay (the time interval from when one stream is selected by a user util the playback starts) and (3) the average playback time lags between peers due to the deployment of the buffering mechanisms. This study should be performed depending of the cluster size.

This section presents a analitical model of the P2PSP using the stochastic differential equation approach [R. Brockett, "Stochastic control," Harvard University, lecure notes.].

Let us make the same fluid assumption as in [13], that is let the content from the server be infinitely divisible and the file be infinitely large so that the server continuously send content to the peers.

6 Definitions

1. Q: The cluster size. The number of peers in the cluster.

2. C: The chunk rate. The number of chunks per second.

6.1 Splitter delivery period

The splitter sends chunks to peers following the Round-Robing scheme. The time T elapsed between two consecutive splitter deliveries depends on the cluster size Q and the chunk rate C, as follows:

$$T = C\frac{1}{Q}. \tag{14}$$

6.2 Steady-state performance

Throughput

The throughput of the system is defined as the amount of content received by all peers per unit time.

Network degree

Lets asume that at time t there are ~~$P(t)$~~ $\#C(t)$ peers in the ~~team~~ cluster. Because each peer sends data to the rest of peers of the ~~team~~ cluster, every peer has a degree of ~~$P(t)$~~ $\#C(t)$.

Bit rates

When the network is stable, i.e. ~~$d\overline{P(t)} = 0$~~ $d\overline{\#C(t)} = 0$ where $\bar{\cdot}$ denotes the expectation operator, the average downloading and uploading rates of peers are ~~b~~ Ya usado antes bs bits/second, which is also the bit-rate of the stream. Peers that are not ~~a~~ be able to send or receive at ~~b~~ bs are rejected from the ~~team~~ cluster. Notice also that ~~b~~ bs is not a constant in most streams. We will express this idea using ~~$b(t)$~~ $bs(t)$ in our argumentation.

Protocol overhead

A comparison between a the P2PSP and a pure server/client architecture?

Network diameter

The diameter is a key property of any overlay structure. It specifies the maximum number of hops necessary to reach any peer from any other peer.

In the P2PSP, the network diameter in a ~~team~~ cluster is a constant equal to 2.

Network delay

Network delay depends on the physical link delay and the available bandwidth (including the queue delays caused by the routing). This second parameter is strongly related to the network load, and therefore, the network delay is ~~a~~ time-varying ~~amount of time~~. ~~Moreover, because the network diameter in the P2PSP is 2, the network delay must to be taken into account twice. Let us denote this time by $N(t) > 0$.~~ Let us denote $Nd_i(t) > 0$ the network delay at physical link i. Due to the diameter of the P2PSP an upper bound of the network delay is $Nd(t) \leq 2\max_i\{Nd_i(t)\}$.

Block rate

The stream ~~S~~ St is divided into an infinite number of orthogonal? blocks ~~S_i~~ St_i of size α such as $St = \bigcup_{i=1}^{\infty} St_i$, where $St_i \cap St_j = \emptyset$. The block rate is defined by

$$Bs(t) = \frac{\alpha}{bs(t)}. \tag{15}$$

Buffering delay

Peers wait until to receive ~~N~~ F blocks to start serving clients (its players). Therefore, the buffering delay at peers can be expressed as

$$Pd(F,t) = \frac{F}{Bs(t)}. \tag{16}$$

Client delay

Most clients (players) use an internal buffer to deal with the jitter. Let's suppose ~~that the time that the client needs~~ the time needed to fill this buffer , i.e., initial delay for the client is $Cd(t) > 0$.

Start-up delay

Also called the playback delay, is the total delay ~~that a~~ an user experiments and can be calculated as the sum of network, buffering and client delays

$$D(F,t) = Nd(t) + Pd(F,t) + Cd(t). \tag{17}$$

6.3 Chunk lost-rate produced by churn and unwarned-churn

Polite and warned churn should never produce a loss of chunks. On the other hand, unwarned churn can produce this lost if the buffer size is too small or if the splitter is congested (remind that the splitter resend those chunks that are requested at least by the half of the peers of the cluster and this could congest its upload link). Independtly of the motive, the rest of the peers of the cluster will lost those chunks that were sent to the outgoing peer while the time that the splitter were not aware of this fact. Under a reasonable network performance, the this time should be smaller than the splitter delivery period T and the number of lost chunks per peer should be only one.

Leo: Juan Alvaro usa una distribucion distinta a la Poisson. ... in the continuity index (ratio of received blocks vs sent blocks by the source that is perfect when reach 1) depending on the arrival and departure rate. ... and in the start-up delay.

The user-driven dynamics of peer participation, or churn, must be taken into account in both the design and evaluation of any large scale P2P application.

Usually, peers (users) arrive and leave acording to a Poisson process with rate λ, i.e., $1/\lambda$ is the average time a peer stay in the team). The Poison counter satisfies that

$$dP(t) = \begin{cases} 1 & \text{at Poisson arrival/departure} \\ 0 & \text{elsewhere} \end{cases} \tag{18}$$

and

$$\overline{dN(t)} = \lambda dt, \tag{19}$$

6.4 Flash crowd dealing (maybe included in the previous section?)

In a flash crowd scenario, large number of peers join the system in a short period of time.

6.5 Maximun free-riding ratio

6.6 DoS impact?

6.7 Scalability

6.8 Are peers synchronized?

In other words, are the peers using the same receiving position?

7 Configurations

This section is devoted to describe a collection of P2PSP configuration examples that could be useful in some specific contexts.

7.1 Trusted peers

Peers are usually run by users that want to play the media. However, one or more trusted peers could be executed by the cluster administrator in order to monitorize the streaming performance and/or minimize the impact of some attacks. Considering that, in general, the trusted peers identities will remain unknown for other peers, the administrator could connect a player to a trusted peer and if the playback is correct, the probability that some malicious peer could be poisoning the media is very low (this could be also done using the monitor peer but remember that the identity of this peer is known and the attacker will avoid to poison the chunks sent to the monitor peer). Another advantage of using monitor peers is that their complaining messages are always true. This permits a effective way of identifying those malicious peers that try to perform a fake complaining attack.

7.2 Super-peers

A super-peer is formed by the union of a peer and an relay server. As can be seen in Figure **??**, the super-peer feeds its team using the P2PSP and one or more different teams by means of the C/S paradigm.

7.3 Premium peers

Premium users pay to view although do not contribute to the team. Their configuration is quite simple because we need to connect directly the premium user's player to a content provider (a origin node). Notice that this does not implies the use of the P2PSP.

7.4 QoS preservation

Is the responsibility of the content provider the maintain a minimum quality of service during the streaming process. A way to achieve this goal is by minimizing the size of the teams, i.e, sharing the streaming load between several splitters up to spend the available bandwidth in the service side.

7.5 Clustering in private networks

In many situations, peers run in hosts of a private networks (see the "Before" part of Figure 4). However, if two or more users that are in the same private network want to play different players, a simple and efficient solution consist in creating a private team using the peer that belong to the public team as a source fo the

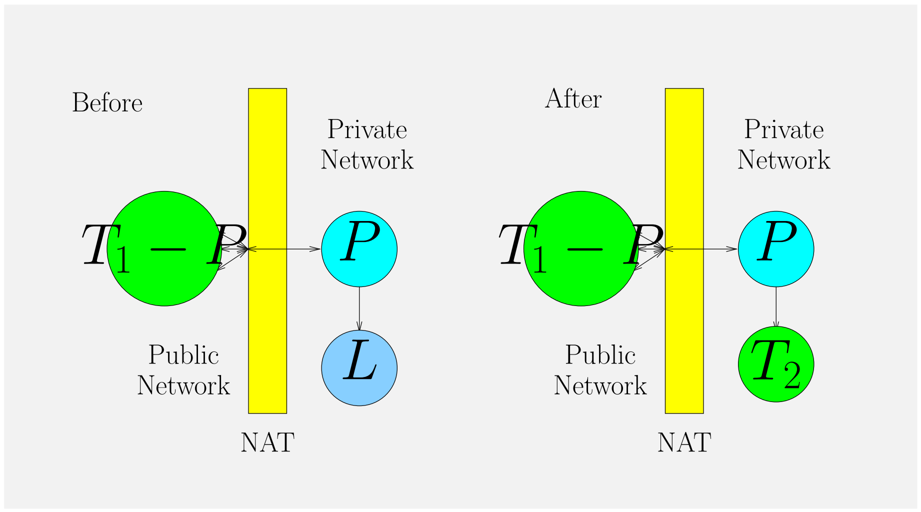

Figure 4: A private clustering scenario. In order to accomodate several private peers, a private team T_2 can be created using the stream produced by the private peer that belongs to the public team T_1.

private splitter (see the "After" part of Figure 4).[13] Note that, as IP multicasting is available on the private network, the private team should be configured to take advantage of this fact.

Similarly as it happens with transparent Web proxies, this locality-aware clustering concept could be also used by ISPs in order to minimize inter-ISP traffic. In a nutshell, if a ISP detecs that there is P2PSP traffic between "local" peers and "foreign" peers, the ISP could deploy a private team. This can be configured by filtering the list of peers that the splitter sends to the peers. This filtering should be performed depending on the locality of the IP addresses of the peers.

7.6 Dealing with symmetric NATs

Whereas cone NATs are used in small and domestic private networks, in most of the corporative LANs the NATs are symmetric. When a peer P is behind a symmetric NAT, only can receive blocks from a public node if P has sent at least a block to it. Another drawback of using symmetric NATs is that each different "connection"[14] has a different public endpoint. Besides this problem, some networks administrator

[13]The reader could think that running another peer inside of the private network the same effect could be achieved. Nonetheless, this is not true because peers know each other by means of a public NAT end-point and most NATs does not allow to communicate private processes using their public end-points.

[14]Using UDP the concept of connection does not make sense. However, we can use this name to reefer to the tuple ((public IP address, public port), (private IP address, private port)), where the public endpoint points to a public peer and the private endpoint points to the private peer.

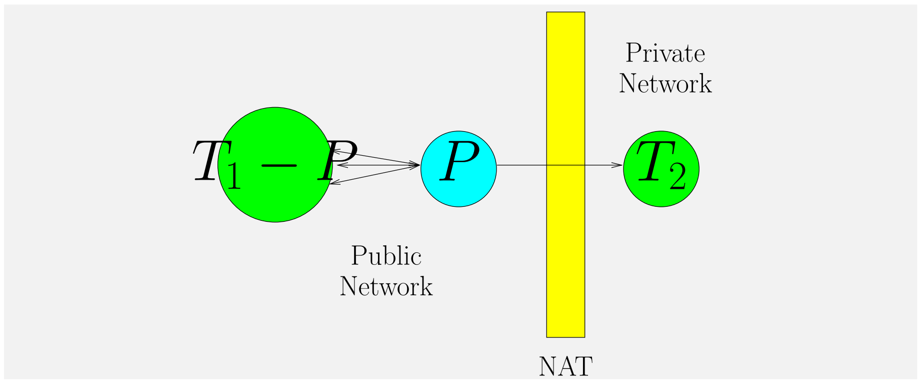

Figure 5: A solution to cross symmetric NATs. A public peer P serves a stream to the private team T_2.

refuses UDP traffic because it can congest their networks easier than using TCP traffic.

One way of solving this problem is creating a different team for the private network, in a similar way as was explained in Section 7.5, but using a peer located in the public network (see Figure 5). Notice that in this configuration, only a TCP connection need to be established through the NAT.

7.7 Building large P2PSP overlays

Although a P2PSP team can scale easily, there are several reasons why it might be desirable to split a large team into smaller ones. For example, if a team fails (for example, the splitter stop sending blocks to the peers), other teams would not be affected by this issue.

Again, a solution to these problems consists in partitioning the big team into a collection of smaller ones. The simplest configuration makes use of the concurrent service that most of streaming servers can perform (see the schema at the left in the Figure 6). However, if the source node can not serve multiple streams, we can divide our big team following a tree structure where the root of the tree is the splitter connected to the source and peers are the leafs of the tree (see the schema at right in the Figure 6).

7.8 Streaming of 3D video

3D video can be broadcasted using two parallel teams implementing the MCS (see Section **??**). In one of the teams the left view (for example) of the stereoscopic video is transmitted whilst in the other team, the other view or simply the differences between both views are transmitted. Thus, a user with a 2D display only need to run a peer that belongs to the left-view team and “3D users” need to run two peers.

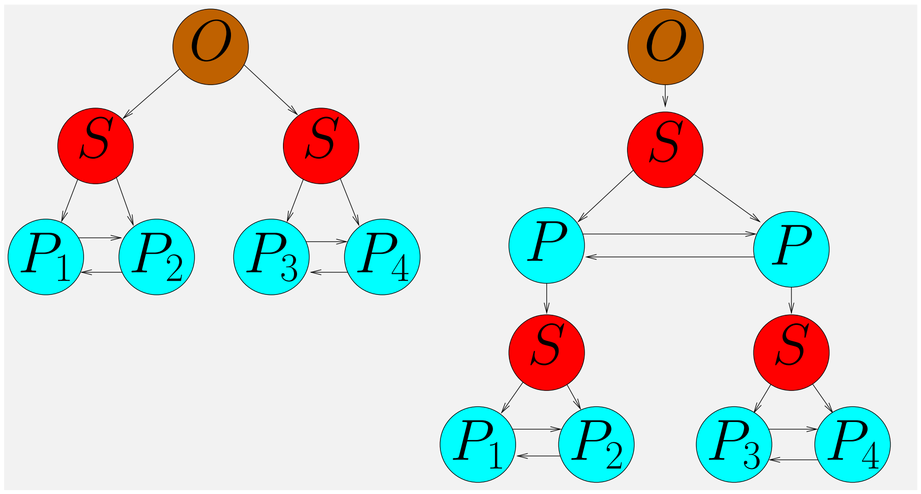

Figure 6: Two examples of an overlay with several teams. On At the left, the concurrency in the source is used to send in parallel the same stream towards two different teams. At the right, concurrency is created by an internal team. In both cases, the sub-teams are smaller than the original one.

In this configuration, the differences-view team should be the channel with lower priority (see Figure 7).

7.9 Simulcast of single-layer media

In some transmission scenarios (such as YouTube), a source can store several copies of the same media, althought variying the temporal resolution, spatial resolution and/or quality. In this situation, each media can be simultaneously broadcasted but in different channels. Thus, peers can switch between teams depending on the variations of the transmission bit-rate, the resolution requested by the user, etc. Notice that, time to perform a switch between channels depends on the buffering time, which depends on the buffer size, the transmission bit-rate and the GOP-rate, that depends on the GOP size and the picture-rate. In a switch, the end of the reception of the old channel should coincide with the beginning of the reception of the new channel, i.e, the buffering time should be predictable. Otherwise, the reception of both channels must be overlapped.

7.10 Streaming of scalable content

The idea introduced in the previous section can be extended to the transmission of scalable video. This type of videos are divided into layers providing spatio-temporal multiresolution and progressive refinement. Moreover, these layers are sortered by priority (for example, in order to get the highest spatial resolution the lower ones

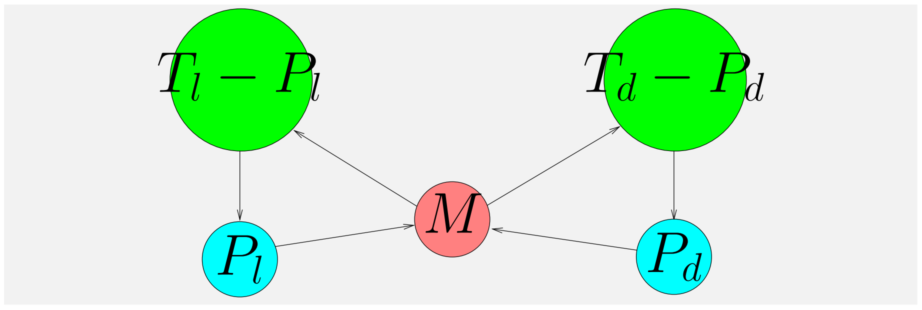

Figure 7: Streaming of 3D video using the ML. The Multi-channel scheduler M prioritizes those incoming blocks that go to the left-view team T_l, in detriment of those ones that go to the differences-view team T_d.

are required first). Therefore, we can transmit each layer in a different channel prioritizing the transmissions using the ML (see Section **??**), in acordance with the priority of the layers and the user preferences.

7.11 Streaming of multiple descripted content

Multiple description codecs provides a set of partially redundant streams so that the quality of the reconstructions improve with the number of descriptions decoded. In this case, if each description is transmitted over a different team, peers can join/left to more/less teams depending on the transmission bit-rate. Moreover, in this case it is not necessary to prioritize the transmission of the descriptions (although a peer could be rejected from a team if it becomes unsupportive).

7.12 Interactive temporal random access (video-on-demand)

Before describing the way in which interactive temporal random access to a video sequence can be done using the P2PSP, it is important to highlight that this possibility represent the worst working scenario of a P2P network because peers tend to retrieve data that it is only interesting for a small amount of them. In the extreme situation where only a peer requests a specific part of the sequence, the bandwidth consumption in the source side is equivalent to the bandwidth consumption in the server side of a CS system.

Keeping this fact in mind, temporal interactivity can be implemented using the P2PSP by configuring a set of teams where each one of them broadcasts a different time-shifted version of the same content. Thus, a peer which wants to move to a different time in the stream, has to leave the current cluster and to move to the team that provides the desired part of the stream.

Obviously, this technique solves the problem of the jumping to a different time in the same video, but does not tackle the problem of dealing with the fast forward and rewind actions. In those situations, it is not necessarily to send all the data of each GOP visited by the peer because probably only one image of the GOP is going to be played. This problem can be tackled by using scalable video media. Having a set of teams where each one of them broadcasts the lowest temporal resolution layer of a different delayed video, fast forward and rewind can be performed when the peer visits only those teams. Finally, notice that temporal scalability does not introduce a significant overhead in the system because its coding overhead is minimal (a video scalable in time needs approximately the same amount of memory than a non-scalable one).

8 Summary

Acknowledgements

This work has been funded by grants from the Spanish Ministry of Science and Innovation (TIN2008-01117, TEC2010-11776-E, TIN2012-37483-C03-03) and Junta de Andalucía (P08-TIC-3518 and P10-TIC-6548, P11-TIC7176), in part financed by the European Regional Development Fund (ERDF) and Campus de Excelencia Internacional Agroalimentario (ceiA3).

References

[1] G. An, D. Gui-Guang, D. Qiong-Hai, and L. Chuang. BulkTree: an overlay network architecture for live media streaming. *Journal of Zhejiang University SCIENCE*, 2006.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, pages 205–217, October 2002.

[3] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *Peer-to-Peer Systems II*, February 2003.

[4] M. Castro, P. Druschel, A-M. Kermarrecand, and A. Rowstron. SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. In *IEEE JSAC*, volume 8, pages 100–110, October 2002.

[5] Y. Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, 2000.

[6] Bram Cohen. `http://www.bittorrent.com`, 2001.

[7] H. Deshpande, M. Bawaand, and H. Garcia-Molina. Streaming Live Media over Peers. Technical report, CS-Stanford, 2002.

[8] M. Dobuzhskaya, R. Liu, J. Roewe, and N. Sharma. Zebra: Peer to Peer Multicast For Live Streaming Video. Technical report, Massachusetts Institute of Technology, 2004.

[9] Free Software Foundation. `http://www.gnu.org/licenses/gpl.html`.

[10] The Xiph.org Foundation. Icecast.org. `http://www.icecast.org`.

[11] P. Francis. *Yoid: Extending the Internet Multicast Architecture*. The ICSI Networking and Security Group, http://www.icir.org/yoid/docs/ycHtmlL/htmlRoot.html, April 2000.

[12] IETF. Peer to Peer Streaming Protocol (PPSP). `http://datatracker.ietf.org/wg/ppsp/charter/`.

[13] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and Jr.J.W. OŤoole. Overcast: Reliable multicasting with an overlay network. In *Operating Systems Design and Implementation*, pages 197–212, 2000.

[14] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *ACM SOSP*, October.

[15] J. Lin, K. Huang, F. Lai, and H. Lee. Secure and efficient group key management with shared key derivation. *Comput. Stand. Inter.*, 31(1):192 – 208, 2009.

[16] Cristobal Medina-López, J.A.M. Naranjo, Juan Pablo García-Ortiz, L. G. Casado, and Vicente González-Ruiz. Execution of the P2PSP protocol in parallel environments. In Guillermo Botella y Alberto A. Del Barrio Garcia, editor, *Actas XXIV Jornadas de Paralelismo*, pages 216–221, Madrid, Septiembre 2013.

[17] J.J.D. Mol, Dick Epema, and Henk J. Sips. The Orchard Algorithm: P2P Multicasting without Free-riding.

[18] J. A. M. Naranjo, L. G. Casado, and J. A. López-Ramos. Group oriented renewal of secrets and its application to secure multicast. *Journal of Information Science and Engineering*, 27(4):1303–1313, july 2011.

[19] V.N. Padmanabhan, H.J. Wang, and P.A. Chou. Resilient Peer-to-Peer Streaming. In *Network Protocols, IEEE International Conference on*, pages 16–27.

[20] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and David E. Culler. SPINS: security protocols for sensor networks. *Wirel. Netw.*, 8:521–534, 2002.

[21] F. Pianese, J. Keller, and E. W. Biersack. PULSE, a Flexible P2P Live Streaming System, booktitle = INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings. pages 1–6, 2006.

[22] The P2PSP Team. Implementation of the P2PSP (Peer to Peer Straightforward Protocol) in Lauchpad. `https://launchpad.net/p2psp`.

[23] The P2PSP Team. Peer to Peer Straightforward Protocol. `http://p2psp.org/en/p2psp-protocol`.

[24] D.A. Tran, K.A. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, volume 2, pages 1283–1292, April 2003.

[25] Lihao Xu and Cheng Huang. Computation-efficient multicast key distribution. *IEEE Trans. Parallel Distrib. Syst.*, 19(5):577–587, May 2008.

[26] Y.Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, pages 1–12, 2000.

[27] Eun-Jun Yoon and Kee-Young Yoo. A secure broadcasting cryptosystem and its application to grid computing. *Future Generation Computer Systems*, 27(5):620 – 626, 2011.

[28] Z. Zhou and D. Huang. An optimal key distribution scheme for secure multicast group communication. In *INFOCOM'10*, pages 331–335, 2010.