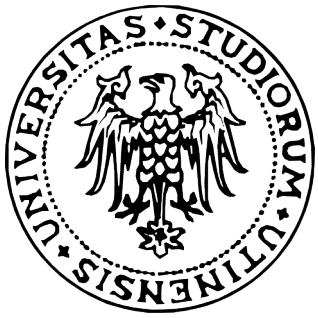


UNIVERSITÁ DEGLI STUDI DI UDINE

Dipartimento di Ingegneria Elettrica Gestionale e Meccanica Dottorato in Ingegneria Industriale e dell'Informazione
XXIV

Tesi di Dottorato

Techniques for Efficient Peer-To-Peer Streaming

Relatore:

Prof. Riccardo Bernardini

Co-Relatore:

Prof. Roberto Rinaldo

Dottorando:

Ing. Roberto Cesco Fabbro

Coordinatore:

Prof. Roberto Rinaldo

To my mother

Abstract

This thesis presents the Peer-To-Peer Epi-Transport Protocol (PPETP). In its early stage PPETP initially was known as Peer-To-Peer protocol, used for the transmission of live streaming over Internet to a large number of residential users, typically, with asymmetric and limited upload bandwidth. It then evolved into a more generic protocol and nowadays it could be considered what is called a multicast overlay protocol. In this thesis all the actual features of this protocol including the technician and practical motivations considered in its construction, are presented. Together with the main characteristics of PPETP, other functionalities that allow the creation of a complete library for the PPETP utilization are also presented. The main topics that will be addressed are the specified functionality for the configuration of the protocol, for peers reachability and security issues. As will be clear during the lecture of this thesis, the main feature of PPETP, at this time, is the utilization of Network Coding (NC) procedures as main coding technique.

To justify this choice, before the presentation of the protocol, two studies are indroduced. The first shows how an opportune use of the NC allows for a reduction of the delay jitter. This characteristic allows an increment of the performance of the applications that utilize PPETP (or more in general of all the applications that utilize the NC).

The latter study concerns the packet loss probability always found in a Peer-To-Peer (P2P) network implementing the NC. This is obviously an extremely important parameter in all network applications using an unreliable transmission protocol, but while in unicast transmissions, this parameter can be "easily" predicted, in a P2P network there are many parameters, often random, that can affect this probability. This study shows some asymptotic results and bounds that can provide useful assistance in the project of developing P2P networks.

Contents

Acronyms

ABNF Augmented Backus-Naur Form

ADSL Asymmetric Digital Subscriber Line

CEP Connection Establishment Procedures

CDN Content Delivery Networks

DAG Direct Acyclic Graph

DCCP Datagram Congestion Control Protocol

DHT Distributed Hash Table

DoS Denial of Service

FEC Forward Error Correction

JSON JavaScript Object Notation

HMAC Keyed-Hashing for Message Authentication

iid Independent and Identically Distributed

ICE Interactive Connectivity Establishment

IETF Internet Engineering Task Force

LC Layered (or Scalable) Coding

LRD Long-Range Dependency

MDC Multiple Description Coding

NAT Network Address Translator

NC Network Coding

NTP Network Time Protocol

PPETP Peer-To-Peer Epi-Transport Protocol

P2P Peer-To-Peer

QoE Quality of Experience

RFC Request for Comments

RTP Real-Time Transport Protocol

RTCP RTP Control Protocol

SDP Session Description Protocol

SRD Short-Range Dependency

TFRC TCP Friendly Rate Control

TLV Type Length Value

Introduction

At present, the transmission of video content over the Internet has assumed a key role in the total network traffic. In fact it is estimated that about 26% of all network traffic is due to on-line video content compared to about 30% for data and 25% for Peer-To-Peer (P2P) traffic [1]. With the general lowering of the residential bandwidth cost, it is not surprising that users have turned their attention towards this kind of content, unavailable until a short time ago. An addition to incrising badwidth, the evolution in the video coding field has also contributed significantly to the diffusion of video content. With the development of new video codecs as the state-of-the-art H.264/AVC standard, with the bandwith available it is now possible to view high quality video at user's homes.

With these technologies, a number of Content Providers normally active in traditional television systems have taken the opportunity to diffuse their contents over the Internet, providing a considerable number of services such as video-on-demand and live transmissions of their channels.

However, although there may be not particular problems for the users with the bandwidth and computational power required for video consumption, the same cannot be said for the content providers. This is evident with very popular contents that have a very high number of users attempting to access them simultaneously. For example, even if a high performance video codec like H.264/AVC is used, the access to elementary 300 Kbit/s streams requires a server bandwidth of 300 Mbit/s for every 1000 users.

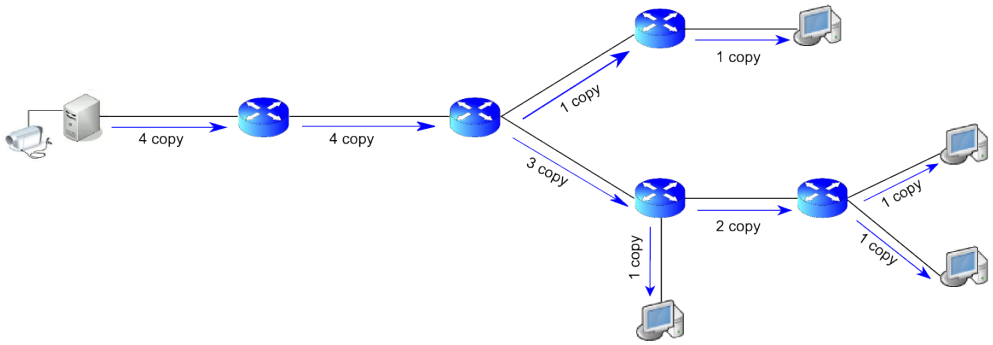
A trivial solution is to use a multi-server architecture where replicas of the same content are made on all the servers, as with the case of the Content Delivery Networks (CDN) [2, 3]. In this example the users are shared amongst the servers. Generally, the servers are located in strategic points (geographically and administratively) within the network. An accurate assignment of the most appropriate server available to the user allows the content-provider to reduce the costs and to increase the quality of the service [1] [4]. The scalability of this system is simply proportional to the number of the available servers of the CDN [2]. The use of a large number of servers certainly leads to very high costs, but nonetheless is still a very widly used system [5, 6, 7] because it can be used not only for streaming applications but for every other service available via the Internet. Due to the numerous contents replicas, CDN are particularly robust to Denial of Service (DoS) attacks because in order to damage the service functionality, it needs attack contemporary a very large number of servers, before the control system detects and reacts to the attack.

Normally, the traffic that travels over Internet is of type unicast that is called a point-to-point protocol; both the hosts of the communication are associated with a public unicast IP address identified in the network [3]. A particular type of IP addresses exist (the class D with addresses in the range [224-239].x.x.x [8]) that is not used to identify a single host, but rather an arbitrary number of these, aggregated in the

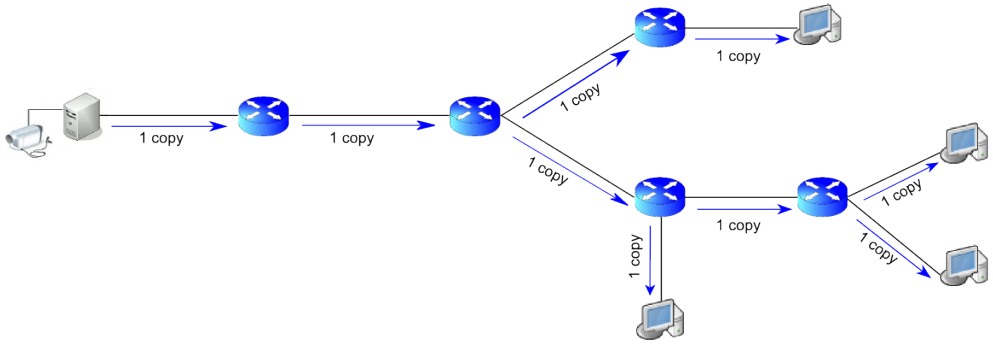
[1] It reduces the costs because strategic point positioning and an accurate choice for the user decreases the use of intra-ISP links and increases the quality of the stream in terms of delay, jitter and packets probability loss.

[3] In reality the association could be not so easy, for example for the presence of Network Address Translator (NAT)

same group [9]. To join the group, a host utilizes the interested multicast address to inform the network (its router) if its intention to join the group. After this phase, the host is inserted in the group and is then ready to receive the streams packets. The transport model used by the IP multicast is one-to-many, which means that data flow is distributed from one source toward many receivers. The main difference between unicast and multicast is that in the latter case, the source produces only one flow and are the routers of the network that duplicate it as needed, toward the others subnets up to reach the hosts, in this way creating a flow tree where the server is the root, and the leaves are the hosts. In the Fig. 1 the difference between unicast and multicast is shown graphically: in Fig. 1(a) it is shown the unicast case where it is visible how the server must produce a stream for every client and how these streams go throughout the entire network. In Fig. 1(b) the multicast is shown; and the efficiency of this scheme is evident, because the links are through by only one stream. It is clear that multicast optimizes the traffic



(a) Unicast



(b) Multicast

Figure 1: Path of a stream with and without the Multicast IP.

both on the network and on the server. The transport protocol that must be used with multicast IP is similar to UDP because of the unidirectionality of the multicast. As with UDP unicast, the IP multicast is subject to packet's loss and duplication. The loss is typical of the IP networks, while duplications are usually due to redundant links between routers. It should also be noted that it is possible for multicast to be congested or have limited bandwidth links, and in these cases losses are possible.

Recently, the use of P2P networks for live streaming has attracted interest in the research community [10, 11, 12, 13, 14, 15, 16, 17]. These networks allow the diffusion of a multimedia stream to a large number of users in a more scalable way, while respecting client-server or CDN technologies. In the P2P approach, every new client, called also *peer*, share its resources (mainly the upload bandwidth) to provide the stream to new clients. If the peer A feed another peer B it is said that A is an *upper-peer* of

B and that B is a *lower-peer* of A (Fig. 2). The bandwidth the server must provide is about equal to the difference between the total download bandwidth (stream's bandwidth times the number of peers) and the total upload bandwidth of the peers:

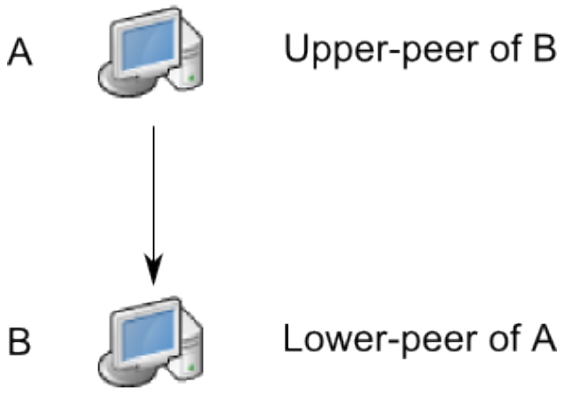


Figure 2: Concept of Upper and Lower peer.

$$DW_{Server} \simeq \sum_{Peers} DW_{Peers} - \sum_{Peers} UP_{Peers}$$

There are a number of problems associated with transmitting a stream over P2P networks as the asymmetry of the users bandwidth, particularly when the upload bandwidth is not enough to retransmit the entire stream, the *churn* that is the evanescence of the peers that could join or leave the network at any time, the presence of networks devices like firewall and Network Address Translator (NAT) [18] that make the exchange of informations with the client problematic. For the first two problems listed above, several of techniques have been developed for working on the network structures and on the coding of the information that is transmitted on those networks. Concerning the structure, the P2P networks for video delivering can be classified, at first glance, as *structured* and *unstructured*: in the first category it is possible to identify a pattern of the stream flow between the peers; the most well-known are tree and multi-tree structures, where the stream starts from the root and travels toward the leaves. [19] considers the case of a single transmission tree and analyzes the upload capacities of the peers. Multi-tree overlays were considered in [20, 21, 22] where it is shown that this type of network has a "phase transition" behaviour if Forward Error Correction (FEC) are employed.

In the latter category, the unstructured (or *mesh*), there is no particular streams flow; the peers exchange informations with each other about the pieces of stream that they have, called *buffer-maps*, and then following a scheduling algorithm, those pieces of stream are exchanged among the peers.

Many works in the literature consider the case of a mesh network based on *pull* approach [17, 23, 24]. P2P networks of this type are fairly common and are typically based on a BitTorrent-like approach: the multimedia content is split into *chunks* that are exchanged among the peers. The other type of approach (the *push*) is more similar to multicast where the data flow is pushed through an overlay network (typically, but not necessarily, organized as multiple tree) [13, 14, 15, 16, 25].

The problem of estimating the performance of P2P networks is very important because it allows to construct a P2P network according to the projected needs. Because of the number of possible approaches to P2P streaming, and of the number of possible performance indicators, literature about the performance of P2P networks is quite variegated. For example, some works consider the delay in a P2P networks and its impact on scalability and data loss [26, 27]; others consider the effect of *churn* and channel changes [28, 29, 30]. Finally, the most recent study [31] considers the effect of packet losses

and node churn on the availability of packets at network nodes. The network model considered in [31] is a multiple-tree overlay multicast, where packets are distributed in round-robin fashion among the different trees.

The multimedia coding technique is also very important for the performance of a video streaming P2P system, and it must be chosen carefully. As said, a P2P system is affected by churn, which means that are possible losses of packets until the structure is restored. For this the stream should be provided with systems to reduce the impact of these losses. For example, in a motion estimated coding, if an infra frame is lost it is not possible to reconstruct the sequence until the next intra frame. In this situation a codec used for network streaming should insert, if possible, frequently infra-frames to reduce the impact of losses. However, this comes at the price of a decrease of the compression efficiency, as the scheme reported in [32, 33].

Two of the most well-known techniques of video coding, particularly used for adapting for video for video streaming, are the Layered (or Scalable) Coding (LC) and the Multiple Description Coding (MDC).

LC consists of the subdivision of the stream into hierarchical importance *layers*. It is composed of a base layer, which is the most important and has the lowest quality, and by some enhancement layers. To reconstruct the original stream, it is necessary to receive at least the base layer, while the reception of the others layers adds to the total quality of the stream. It is important to remember that the enhancement layers are hierarchical and all the previous layers must therefore be received before a successive layer can be obtained for enhancement. A simple example could explain easily this concept; let call B the base layer and $L_1, \ldots, L_N$ the Ns enhancement layers. If it wants to obtain a full quality video it must receive the base layer B and all the enhancement layers L_i with $i = 1, \ldots, N$, while the lower quality video is composed only by the B layer. All others quality levels are composed by B and L_i with $i = 1, \ldots, K$ with $K < N$. If it receives B and $L_1, L_2, L_4, \ldots, L_N$ the total quality is the same that it is possible to obtain receiving only B and L_1, L_2 because it doesn't receive the layer L_3; a lack is not permitted. Often, because of its importance, the base layer is more protected from losses than the other layers, because its correct reception is essential to reconstructing the video. In this way, a stream can always be reconstructed, even if it is with a lower quality. A scheme of this procedure is presented in Fig. 3 and a detailed description is reported in [34]. In this figure it is visible as a single encoder that produces all the layers, while the decoders, are able to obtain a certain quality according to the layers received, shown on their left. The first and the fourth decoders reconstruct, respectively,

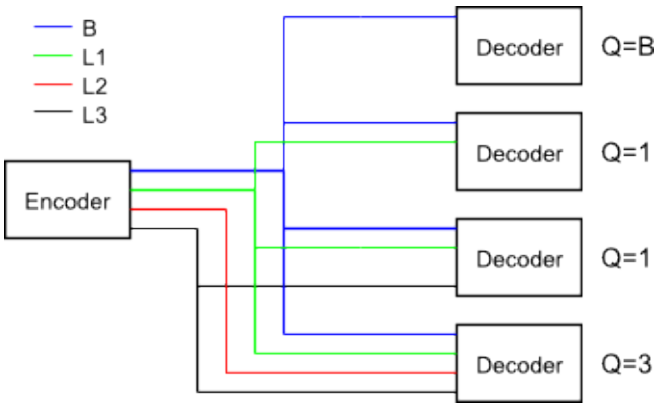


Figure 3: Layered Coding scheme.

the lowest video quality and the full video quality, because the first receives only the base layer and the fourth receives all the layers. The second and the third decoder can reconstruct a video with the quality corresponding to the enhancement layer L_1: the first because it receives only the base layer and the first enhancement layer L_1, while the latter, even if it receives the layer L_3, cannot reconstruct a full quality video because it does not receive the enhancement layer L_2.

MDC is another kind of encoding that as the LC subdivide the stream in more sub-streams or *descriptions*. Different to LC, the sub-streams have all the same importance, so it is not the progressive reception of all the layers that is important, but the total quality of the reconstructed stream is proportional to the number of the received descriptions. In this case a scheme of this concept is also reported in Fig. 4 and a more detailed description of this technique is described in [35]. This coding naturally

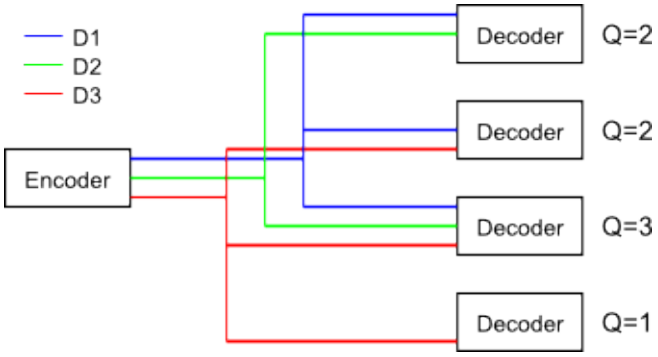


Figure 4: Multiple Description Coding scheme.

adapts to multi-tree networks [36, 37], in fact each description could be sent on a tree where a user is connected to one or more trees receiving a different quality stream. It is not important which tree is connected, but how how many trees it is connected to. This allows the server to code a video only once, and then distribute it with a multi-resolution capability.

There is another important coding technique widely used with P2P networks called Network Coding (NC) [38, 39]. It is very different from the other techniques seen before because it doesn't encode directly the frames of the video, instead works at stream level [4]. The stream, essentially, is a bits sequence generated by an encoder that is considered such a sequence of elements of a finite field.

The encoding procedure, called also *reduction procedure*, is done in three steps: the first is the creation of the *reduction vector* $\mathbf{r} = [\alpha_1, \alpha_2, \ldots, \alpha_R]$ of R elements of the finite field. R is called *reduction factor*, because as will be explained shortly, the data volume resulting by the encoding procedure is about R times smaller than the original data. The data to be encoded is organized as a matrix $\mathbf{C}$ called *Content data (or content packet)* of R rows and any number of columns.

$$\mathbf{C} = \begin{bmatrix} P_1 & P_{R+1} & \ldots \\ P_2 & P_{R+2} & \ldots \\ \vdots & \vdots & \\ P_R & P_{2R} & \ldots \end{bmatrix} \tag{1}$$

The encoding procedure is the left multiplication of the reduction vector times the content packet to obtain the *reduced data (or reduced packet)* $\mathbf{u}$:

$$\mathbf{u} = \mathbf{rC} \tag{2}$$

[4]The stream could be produced by any kind of source.

For the decoding or *reconstruction* of the Content Packet, at least R reduced packet $\mathbf{u_i}$ are necessary, obtained from linearly independent reduction vector $\mathbf{r_i}$. With these is possible to construct the following system

$$\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_R \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_R \end{bmatrix} \mathbf{C} \tag{3}$$

that written in matricial form become

$$\mathbf{U} = \mathbf{RC} \tag{4}$$

Because of the linear independence of the reduction vectors, the matrix $\mathbf{R}$ can be inverted, and this allows to retrieve the original data packet

$$\mathbf{C} = \mathbf{R}^{-1}\mathbf{U} \tag{5}$$

A method for using this technique to distribute a video over a P2P network is reported in [40].

A very interesting characteristic of the NC is the versatility and the number of possibilities created by this technique. Differently from MDC and LC, network coding is not designed to encode only video, but it is suitable to encode data of any kind. In fact, data is seen as an element of a finite field independent of its meaning, in this way encrypted data can also be coded. The network coding is a lossless encoding which means that the decoded data is exactly equal to the original data.

As indicated in [39] and [40] it is possible to achieve a more robust system based on NC using redundant data, that is obtained using more reduced packets then the minimum necessary for the reconstruction, such as in FEC systems. It is possible to collect more than R reduced packets, obtaining the redundant system

$$\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_R \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_R \\ \vdots \\ \mathbf{r}_N \end{bmatrix} \mathbf{C} \tag{6}$$

Not all the equations of this system are necessary in order to obtain the Content Packet again, for this reason some of them are discarded (or can be lost) to obtain a system like that of Eq. (3). This allows corrupted reduced packets, or that obtained from linearly dependent reduction vectors, to be discartded. Moreover, receiving reduced packets from more sources than necessary increases the probability of reconstruction in a lossy network, because, even if pakets are lost, it is probable that the receiver receives enough packets for the reconstruction.

The NC has been adapted particularly to be used on unstructured P2P networks, in fact the NC allows an exchange of reduced packets among peers without the needs of a determined flow; the only requirements are that the coefficients used for the reduction must be different, and that the network should be acyclic. In Fig. 5 is shown a typical use of the network coding in a P2P network. In the figure, a reduction factor of two is used. The source S produces two different version of reduced packets. The peers that cannot reconstruct the original packets[5] (because they receive only one reduced packet) forward the received packet toward the other nodes connected to them. The peers that can reconstruct the content packets reduce them again, producing new reduced packets, and then forward them toward

[5]In the Fig. 5 are drawn as white circles, while the nodes that can reconstruct the packets are white circles with a label inside.

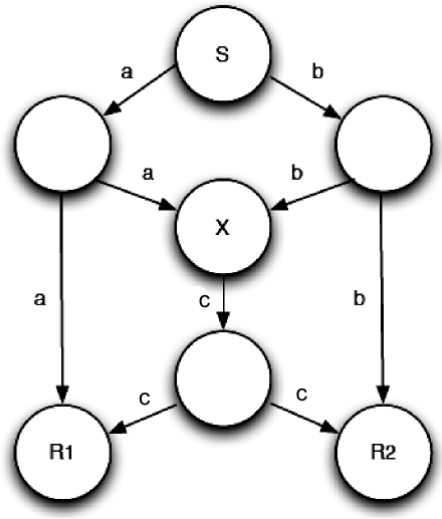


Figure 5: Use of the Network Coding in a P2P network.

the other nodes of the network.

In this thesis, Peer-To-Peer Epi-Transport Protocol (PPETP), an emerging protocol for the transmission of data in a multicast fashion is presented. It is based on a P2P network using NC as main coding technique. Together with PPETP some advanced features for the configuration of it are also shown, for the realization of practical applications. Furthermore, some theoretical and practical demonstrations of the impact of the NC on the characteristics of P2P systems are presented to demonstrate the possibilities created by this technique.

In the Chapter 1 an analysis of the delay jitter on a P2P network using NC is discussed. This is a very important parameter in many applications, especially for video streaming systems. This study proves the possibility of the reduction of this parameter by varying the parameters of the NC.

Together with the jitter, another important parameter is the packet loss probability experienced by a node of a P2P network using NC procedures. The effects of packets loss is particularly critical in P2P environments, because together with the loses experienced within a normal network, extra the losses are added due to the churn. In the Chapter 2 this probability is analyzed asymptotically, on different network structures, to obtain some boundaries to the network size.

After these preliminary studies, the Chapters 3 and 4 present PPETP. In the first of these two chapters, the application of the protocol and its main features are presented, while in the second chapter some more advanced, but nonetheless important characteristics are described. This chapter also is shows a “PPETP-related” configuration protocol that is not mandatory for the functioning of PPETP but rather for its configuration.

The Appendix A shows the main PPETP plugins. As explained in subsequent chapters, some of the features of PPETP are delegated to plugins to make the protocol more expandable for future or specific improvements. This Appendix presents the pre-build plugins that allow for the construction of complete working applications.

Chapter 1

Jitter reduction in P2P networks

One of the most important parameters, in the study of P2P networks for live streaming applications on the Internet, is definitely the *jitter*. This is the variability of the delay spent by packets to traverse the path between the source and the network clients. At the receiver the stream is decoded according to a precise time scheduling and given the uncertainties of the arrival instant of packets from the network is required the interposition of a packet's buffer. Greater is the delay variance and greater must be the size of the buffer (see Section 1.2). This causes a significant delay start up, which is very unlikely to the user that had to wait until the buffer fills up before being able to play the stream. For this reason, the characterization of this parameter is important for the design and optimization of systems for live streaming.

In cases where there is a single source (server) that provides the stream to the clients, the characterization of this parameter depends mainly on the state of the network, and is relatively easy to measure. However interesting results are reported in [41, 42, 43, 44]. These studies reveal the statistical self-similarity (or fractal) nature of the delays on the Internet. In addition, Li and Mills show that this characterization is highly dependent on the interval elapsed between one package and the others. When this interval is short, the network exhibits the so called Short-Range Dependency (SRD), while when this interval is long it refers to Long-Range Dependency (LRD).

In this Section P2P networks that use network coding for streaming are discussed. With NC it is possible, as explained in the Introduction, to utilize a greater number of upper-peer respect the minimum necessary for the reconstruction of the content packet. This is in order to achieve an increase in the robustness and, as it will be shown, a decrease in the mean jitter of the network. Calling N the number of upper-peers and K the minimum number of packets to reconstruct the content packet, the choice of a determinate upper-peer's degree of redundancy (N/K), allows the content packet reconstructions as soon as the first K packets arrive; these packets are sent by the K "faster" upper-peers. This thereby leads to a decrease of the overall delay and jitter. Obviously, increasing the ratio N/K corresponds decrease of the jitter, but at the same time there is an increase of the bandwidth overhead, because $N-K$ packets travel over the network but are not utilized. Therefore it is important to find a compromise based on the desired features of the system.

1.1 Preliminary remarks

1.2 Video buffer

Previously, it was said that in order to counteract the presence of the jitter, normally in the video streaming player, a buffer is intrduced. In this brief section the motivation for this is explained. Considering that the stream server sends packets every nT seconds with T fixed and $n \in \mathbb{N}$ and for simplicity the network delay, corresponding at the n-th packet is the random variable x_n, with Gaussian distribution $\mathcal{N}(m, \sigma^2)$. Supposing the receiver has a buffer of Δ seconds, the playback time of the video packets are $nT + \Delta$. This means that if $nT + x_n > nT + \Delta$ the player will not be able to correctly reproduce the video. From the dimension of the buffer and from the distribution of the delay it is possible to calculate the reconstruction error probability.

$$\begin{aligned} P[err] &= P[nT + x_n > nT + \Delta] \\ &= P[x_n > \Delta] \\ &= P\left[\eta_n > \frac{\Delta - m}{\sigma}\right] \\ &= 1 - \Phi\left(\frac{\Delta - m}{\sigma}\right) \end{aligned} \tag{1.1}$$

More interesting is to calculate the dimension of the buffer to ensure a given error probability ε

$$\begin{aligned} P[loss] \leq \epsilon \Rightarrow 1 - \Phi\left(\frac{\Delta - m}{\sigma}\right) &\leq \epsilon \\ 1 - \epsilon &\leq \Phi\left(\frac{\Delta - m}{\sigma}\right) \\ \Phi^{-1}(1 - \epsilon) &\leq \frac{\Delta - m}{\sigma} \\ \Delta &\geq \sigma\left[\Phi^{-1}(1 - \epsilon)\right] + m \end{aligned} \tag{1.2}$$

clearly highlighting the relation between the jitter σ and the buffer size Δ. From the Eq. (1.2), it seems that the mean of the delay also deeply affects deeply the buffer size. In reality, this can be neglected, as it can only be seen as that the server and the client are not synchronized.

1.2.1 Network

The network model that will be analyzed is reported in Fig. 1.1. In this model, a stream server encodes the stream with network coding and provides at least K different reduced versions toward peers of the P2P networks. A client (or node) $\mathcal{C}$ is connected to $N \geq K$ upper-peers U_n, with $n = 1, \ldots, N$, of the network.

1.2.2 Delay Model

Let's call D the time necessary for a packet, transmitted by the server, to reach the client. This delay can be seen as composed by two elements: the time necessary for the packet to reach the upper-peer, and the time required to travel from the upper-peer to the client. In this section it is considered only the latter delay is considered and particularly its standard deviation σ_D that could be seen as the client's *jitter*.

From the point of view of the delay D analysis, the Fig. 1.1 could be simplified as in Fig. 1.2. In this figure, a packet P leaves the server at the instant t_0 (it is possible to choose $t_0 = 0$ without loss of

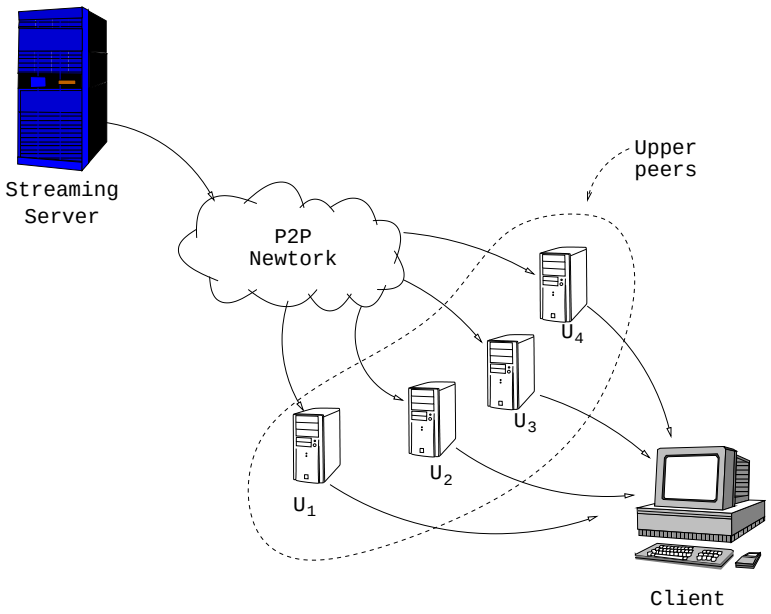


Figure 1.1: Network Model.

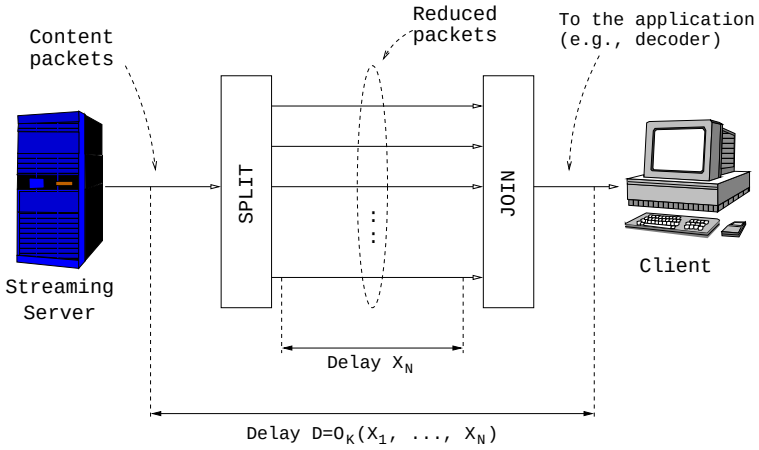


Figure 1.2: Equivalent model.

generality), it is divided into its reduced version $\widehat{P}_n$, produced by the upper-peers U_n, and each reduced version is sent to the client. The time requested at the packet $\widehat{P}_n$ to reach the client is modeled with the random variable X_n, which takes into account the time required for the whole path server $\rightarrow$ peer $U_i \rightarrow$ client.

1.2.2.1 Delay statistical distribution

In the following, it will be supposed that the overall server-client delays X_n, for packets received from the upper-peer U_n, are modeled as independent random variables with conditional distribution $F_n(x) = G_{X|\theta}(x|\theta_n)$ where θ_n is a random parameter belonging to a suitable parameter space Θ and distributed according to probability density f_θ.

Remark 1.1. *More precisely, we will suppose that pairs $(X_n, \theta_n), n = 1, \ldots, N$, are Independent and Identically Distributed (iid) random variables assuming values in $R \times \Theta$, that f_θ is the marginal density of θ_n and that $G_{X|\theta}(x|\theta_n)$ is the conditional distribution of X_n given θ_n.*

This model can account for the fact that the delay of packets arriving through different U_n may exhibit a distribution with, for instance, mean values chosen randomly. In other words, it model the fact that the delays corresponding to different paths can have a given distribution, possibly, for instance, with a different mean.

1.2.3 Order Statistic

The time D at which a packet P is recovered, is equal to the time the K-th reduced packet arrives, that is, $D = O_K(X_1, \ldots, X_N)$, where $O_K : \mathbb{R}^N \to \mathbb{R}$ denotes the function that distributes its arguments in increasing order and returns the K-th. When convenient, it is possible to write $O_K(\mathbf{X})$ instead of $O_K(X_1, \ldots, X_N)$, with $\mathbf{X} = [X_1, \ldots, X_N]$.

1.3 Theoretical Results

1.3.1 Reduction to iid case

The model introduced in Section 1.2.2.1, where random parameters θ_n are allowed for correspond to iid X_n, as claimed by the following property.

Property 1.1. *Let (X_n, θ_n) be as in Section 1.2.2.1. Then $X_n, n = 1, \ldots, N$, are iid random variables with distribution*

$$F_X(x) := \int_\Theta G_{X|\theta}(x|t) f_\theta(t) dt. \tag{1.3}$$

Because of Property 1.1, in the remaining part of this study it will be supposed the delays iid. The delay from the n-th node to the peer will be represented by X_n. Mean m_X and standard deviation σ_X will be named, respectively, *basic delay* and *basic jitter* and will represent the average delay and jitter that a node would experience when receiving data from a randomly chosen peer.

Note that the model above (with distributions depending on a random parameter), is suitable for obtaining the "average performance" through a set of clients. See also [45] for results related to the case where each X_n is distributed according a fixed distribution.

1.3.1.1 Reduction to $m_X = 0, \sigma_X^2 = 1$

Besides reducing the problem of studying D in the iid case, it is also possible to assume, without loss of generality, X_n normalized to zero mean and unit variance. More precisely, let m_X and σ_X^2 be the mean and variance of X_n and let $X_n^o := (X_n - m_X)/\sigma_X$ be the "normalized" version of X_n. It is easy then to prove the following property.

Property 1.2. *Let F_D and F_{D^o} be, respectively, the distributions of D and $D^o := O_K(X_1^o, \ldots, X_N^o)$. The following equality holds*

$$F_D(x) = F_{D^o}\left(\frac{x - m}{\sigma}\right). \tag{1.4}$$

Moreover, if m_D, m_{D^o}, σ_D^2 and $\sigma_{D^o}^2$, denote the mean and variance of D and D^o, the following equalities hold

$$m_D = \sigma_X m_{D^o} + m_X \tag{1.5}$$

$$\sigma_D^2 = \sigma_X^2 \sigma_{D^o}^2. \tag{1.6}$$

Note that from Property 1.2, it follows that the ratio σ_D/σ_X between the jitter with network coding and the basic jitter depends only on N, K and the normalized distribution of X_n. This property allows one to present the results (as shown in Section 1.4) on jitter reduction in a "normalized" form that does not depend on the basic jitter σ_X.

1.3.2 The iid case

From Order Statistics theory [45], the distribution of $O_K(X_1, \ldots, X_N)$ has the expression:

$$F_D(x) = \sum_{j=k}^{N} \binom{N}{j} F_X(x)^j (1 - F_X(x))^{N-j}. \tag{1.7}$$

From (1.11), one can easily obtain the ℓ-th moment of D

$$\begin{aligned} m_D^{(\ell)} &= \mathbb{E}[D^\ell] \\ &= N\binom{N-1}{K-1} \int_{-\infty}^{\infty} u^\ell \cdot F_X^{K-1}(u)\,(1 - F_X(u))^{N-K}\, dF_X(u). \end{aligned} \tag{1.8}$$

By changing variable u in (1.8) with $x = F_X(u)$, one obtains

$$m_D^{(\ell)} = N\binom{N-1}{K-1} \int_0^1 \left[F_X^{-1}(x)\right]^\ell \cdot x^{K-1}(1-x)^{N-K} dx. \tag{1.9}$$

From (1.9), it is possible to calculate the expectation $m_D = m_D^{(1)}$, the statistical power $M_D = m_D^{(2)}$ of D and the jitter $\sigma_D = (M_D - m_D^2)^{1/2}$.

The calculation of the Eq. (1.9) requires the numerical resolution of a complex integral, depending on the inverse cumulative distribution function $\left[F_X^{-1}(x)\right]^\ell$. For the sake of numerical computation, it is possible to estimate (1.9) by approximating $\left[F_X^{-1}(x)\right]^\ell$ with a polynomial

$$\left[F_X^{-1}(x)\right]^\ell \approx \sum_{n=0}^{d} a_{n,\ell} x^n. \tag{1.10}$$

By using (1.10) in (1.9) the following is obtained

$$\begin{aligned} m_D^\ell &\approx \sum_{n=0}^{d} a_{n,\ell} \cdot N\binom{N-1}{K-1} \int_0^1 x^{K+n-1}(1-x)^{N-K} dx \\ &= \sum_{n=0}^{d} a_{n,\ell} b_n^{(N,K)}, \end{aligned} \tag{1.11}$$

where

$$b_n^{(N,K)} = N\binom{N-1}{K-1} \beta(K+n, N-K+1). \tag{1.12}$$

and β denotes the Beta function [46]. For the sake of completeness, in the Tab. 1.1 are reported the coefficients $a_{n,\ell}$, relative to the case, considered in [42], of X_n Gaussian with $m_X = 0$ and $\sigma_X^2 = 1$ (this suffices because of Property 1.2). Moreover, for the Gaussian distribution, it was taken in consideration that $F^{-1}(x)$ is an odd function, hence it is been approximated with an odd polynomial, while $[F^{-1}(x)]^2$ is an even function, therefore the approximating polynomial is also even.

1.4 Experimental results

Three different types of experiments have been executed:

Table 1.1: Polynomials coefficients for the Gaussian case.

	a_0	a_1	a_2	a_3	a_4
$F^{-1}(x)$	3.386	-5.079	-5.030	-1.669	
$[F^{-1}(x)]^2$	19.414	-38.828	35.194	-15.780	2.733

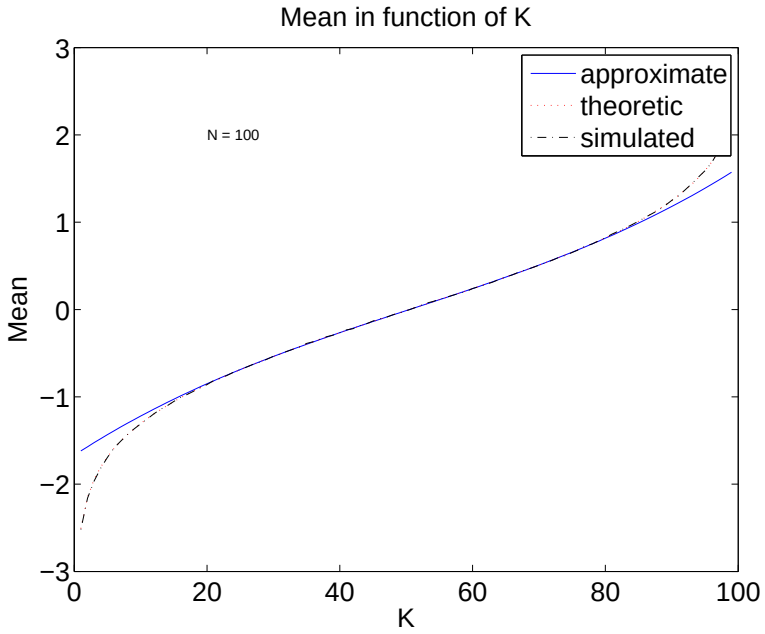


Figure 1.3: Comparison between theoretic, approximate and simulated mean delay for $N = 100$ (peer delays distributed as $\mathcal{N}(0, 1)$.)

1. Verification of the quality of the approximation (1.11) in the Gaussian case (Section 1.4.1).
2. Verification of the theoretical results by means of simulation (Section 1.4.2).
3. Verification of the predicted results against a test carried out on the real Internet, using the Planetlab [47] network (Section 1.4.3).

1.4.1 Approximated moments

The first suite of experiments was aimed at verifying the quality of approximation (1.11) in the Gaussian case for the calculation of the mean and the variance. It was computed the "exact" values of the moments by numerically evaluation of the integral in (1.9) for $N = 100$ and $K = 1, \ldots, N$. The approximate moments were calculated via (1.11) using the coefficient of Tab. 1.1 for the same values of N and K. Both the exact and the approximate results are compared in Figs. 1.3 and 1.4 where it can be seen that the quality of the approximation is well within in the range $K = 20, \ldots, 80$.

1.4.2 Simulations

In this set of experiments, the mean delay m_D and the jitter σ_D was obtained by generating 1000 times a vector of $N = 100$ Gaussian random variables (with zero mean and unit variance) and then calculating the mean and the variance of the resulting K-th order statistics. This procedure was repeated for $K = 1, \ldots, N$. The results are reported in Figs. 1.5 and 1.6. It is interesting to observe, in Figs.

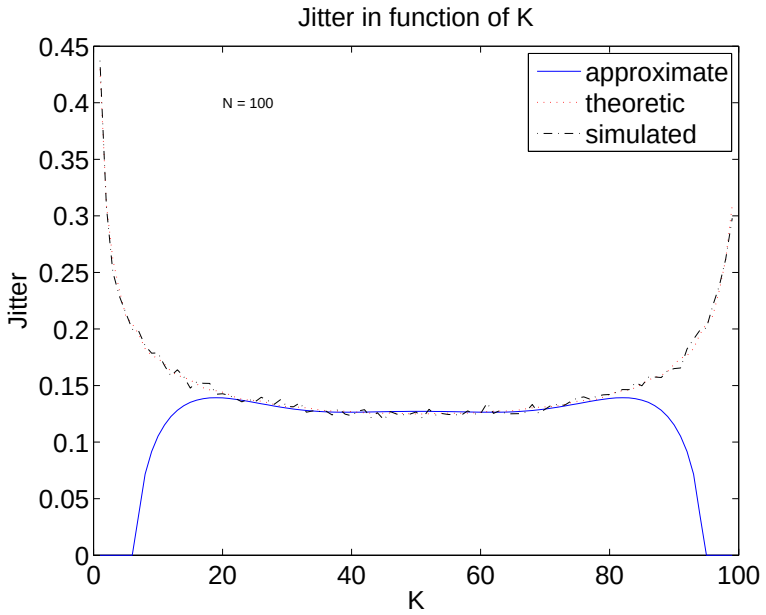


Figure 1.4: Comparison between theoretic, approximate and simulated jitter for $N = 100$ (peer delays distributed as $\mathcal{N}(0, 1)$).

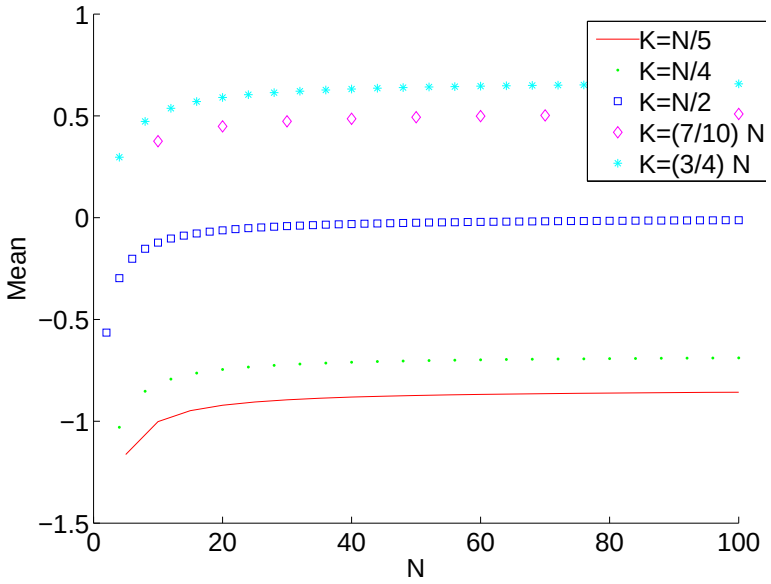


Figure 1.5: Mean delay as function of N (peer delays distributed as $\mathcal{N}(0, 1)$).

1.5 and 1.6, the mean and the standard deviation of D, as a function of N for fixed values of the ratio $\rho = K/N$. Note that the mean and the statistical power of D converge, for large N, to a value that depends only on ρ , while the jitter (the standard deviation of D) goes to zero as $1/\sqrt{N}$ and is almost independent on ρ . Such behaviour is actually predicted by some results of asymptotic theory of order statistics, and it even holds in more general cases (see, for example, Chapter 9 of [45]). This implies that on a first approximation, the expected reduction in jitter depends only on N and not on ρ , or on the exact statistical distribution of the delays.

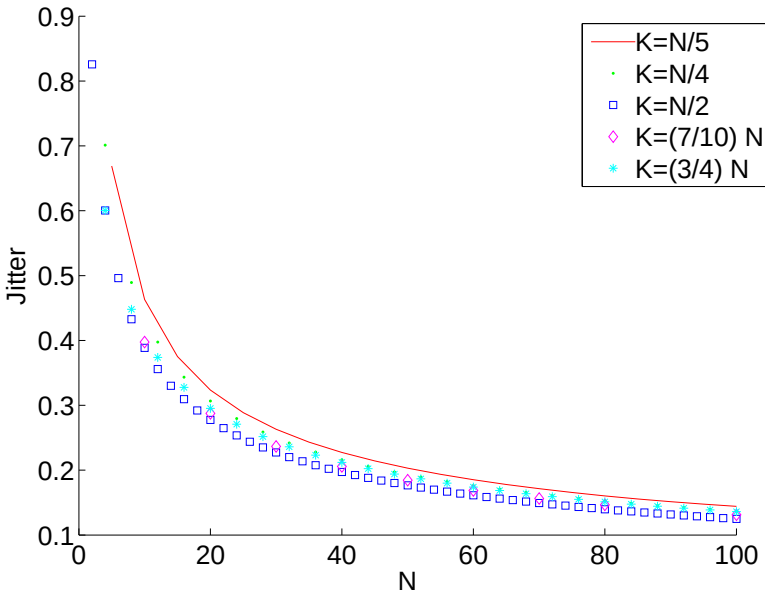


Figure 1.6: Jitter as function of N (peer delays distributed as $\mathcal{N}(0,1)$).

1.4.3 Planetlab test

The objective of the last set of experiments was to validate the theoretical results presented erlier with a test done on the real Internet. The testbed of the experiment was the Planetlab network. A network of 101 nodes, it has been used as such: one node played the role of client, while the others, like upper-peers, sent data to it. Each packet carried a sequence number (that allowed to match packets received from different peers) and a timestamp with the time the packet was sent over the network. With these informations the client could calculate the delay experienced by the packet. The nodes were synchronized by means of Network Time Protocol (NTP) [48], with a synchronization error smaller than 10 ms (comparable to the interrupt frequency of the Linux kernel used by the Planetlab nodes). Each node sent 1000 packets, one packet every 0.5 seconds. Fig. 1.7 shows the histogram of the peer's delay. Note that the nodes can be partitioned into two groups: one group with a very small delay (< 0.06 seconds) and another group with larger delays. The origin of these groups is probably due to the geographical distance between the nodes and/or network conditions. In collecting the results, since jitter is more critical for packets with a large delay, only the second group of packets with delay greater than 0.06 s was considered. Moreover, the error due to the precision of NTP synchronization and the multi-task nature of the system, was of the same order of magnitude than the average delay of the first group, making the data from this group less reliable. The average and the standard deviation associated with the second group of nodes are, respectively, $\overline{m} = 92$ ms and $\overline{\sigma} = 3.8$ ms ($\overline{\sigma}^2 = 14.5 \mu s^2$). The values of the mean delay and jitter of the second group were utilized to predict the mean and the jitter of reconstruction time for $K = 1, \ldots, N$, where now $N = 50$ by approximating the delay distribution with a Gaussian with mean $\overline{m}$ and variance $\overline{\sigma}^2$.

Figs. 1.8 and 1.9 compare the measured mean delay and jitter with the corresponding theoretical prediction. By exploiting Property 1.2, the times on the y axes in Figs. 1.8 and 1.9 are normalized to (measured in units of) $\overline{\sigma}$. Note that, with this time unit, the basic jitter is equal to 1. The agreement between experimental data and theoretical prediction can be considered fairly good, considering the fact that the data of Fig. 1.7 (second group) have only roughly a Gaussian distribution. Note that, for

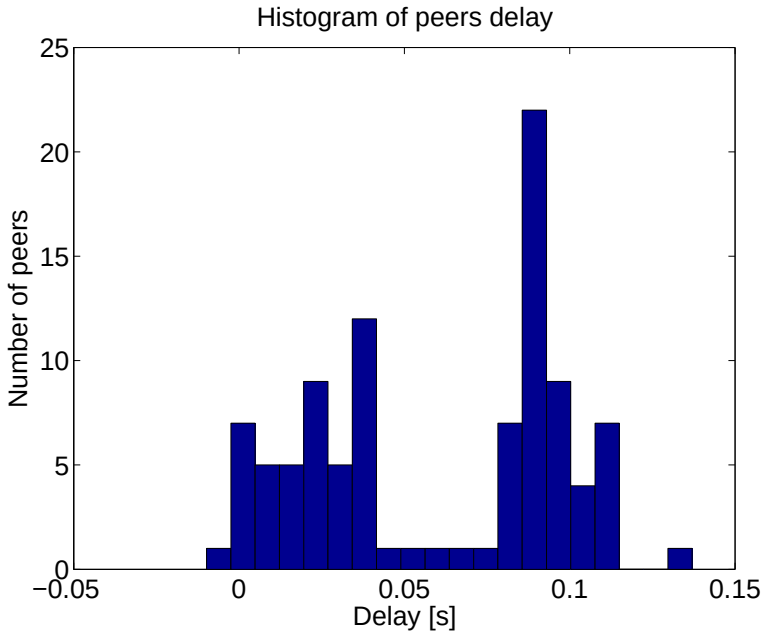


Figure 1.7: Histogram of peer delays.

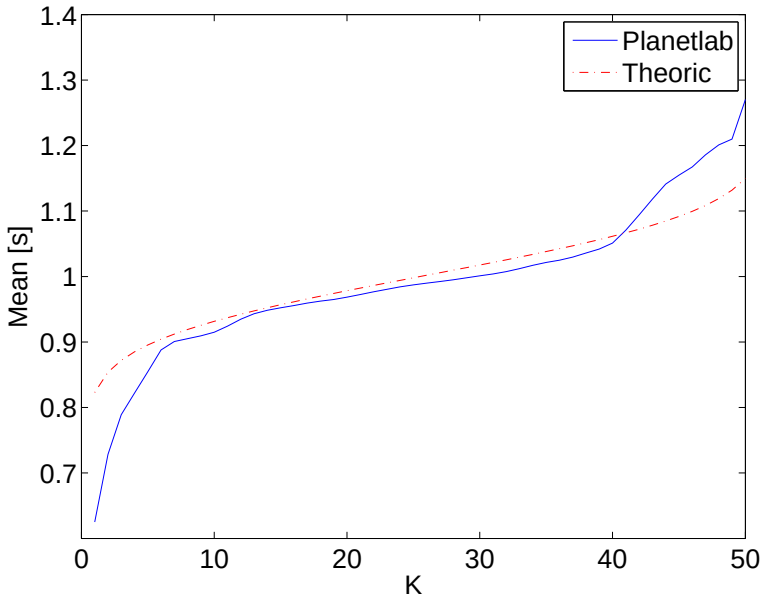


Figure 1.8: Experimental and predicted mean delay as function of K (in units normalized to basic jitter).

$K = 10, \ldots, 40$ Fig. 1.9 shows that with $N = 50$ nodes the jitter is reduced from 1 (in normalized units) to approximately $0.14 \approx 1/\sqrt{N}$, coherently with the results of Section 1.4.2.

1.5 Conclusions

In this chapter the delay jitter experienced by a node that receives data, streamed over a P2P network, that employs network coding procedures was analyzed and explained. The analytical results have been

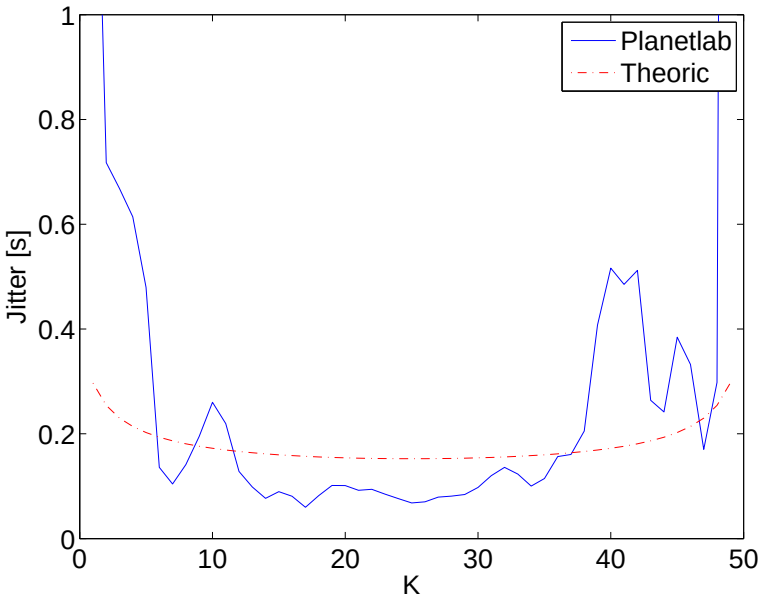


Figure 1.9: Experimental and predicted jitter as function of K (in units normalized to basic jitter).

validated by means of experiments, on the real Internet network, with good agreement. The analytical and experimental results show also that the jitter experienced by the node decreases as $1/\sqrt{N}$, where N is the number of upper peers of the node. This demonstrate that theoretically, it is possible to reduce the jitter arbitrarily, at the cost of an increase in bandwidth.

Chapter 2

Packets Loss Probability in Network Coding enabled P2P networks

2.1 Introduction

This study is interested in the packet loss probability experienced by the application that "sees" the P2P network look like an erasure channel with an equivalent loss probability P_{eq}. The type of network that is analyzed in this work is stream-based and *chunkless*, as in Octoshape [13], Lava [14], PPETP [25] (and Chapter 3), R2 [15] and Splitstream [16]. The model analyzed here is more general than the multiple-tree model analyzed in [31]. One key point is that it allows each node to lower its upload bandwidth by applying *reduction procedures* to the multimedia content (e.g., as the NC explained in the Introduction). The reduction procedure employed, allows nodes with limited upload bandwidth to contribute to transmission, and in some cases, to increase the reliability of the system.

Among the papers cited earlier, [31] is closest to this work, although it uses a different network model and different performance metrics. More precisely, in [31], the authors consider the "global metric" represented by the probability that a randomly chosen node in the network can reconstruct a given packet, while this work is interested in the asymptotic behaviour of P_{eq} as a function of the distance between a node and the server. The main result is shown here is that, although the value of P_{eq} experienced by a node converges to 1 when the distance from the server grows to infinity, it is possible to choose network parameters that make this converge very slow, so that P_{eq} remains negligible in networks of practical size. This is shown by deriving analytical bounds from the P_{eq} convergence rate.

2.2 Notation

In this study some particular symbols to denote different concepts will be used; The symbol $\delta_{a,b}$ is the Kronecker delta, that is, $\delta_{a,b} = 1$ if $a = b$ and $\delta_{a,b} = 0$ otherwise. Markov chains with a finite alphabet are also considered; it is used the symbol $\rightarrow$ to denote a one-step reachability relation, that means it is possible write $a \rightarrow b$ if the chain can transit from a to b in one step. It is used $a \rightarrow^n b$ if there is a path of length n from a to b and $a \rightarrow^* b$ if there is a path of *any* length from a to b. If the Markov chain is homogeneous, it is possible to use the shorthand $P(a \rightarrow b_1 \rightarrow b_2 \rightarrow \dots b_N)$ to denote $P[s_{n+N} = b_n, \dots, s_{n+1} = b_1 | s_n = a]$. Note that this notation factorizes, that is, $P(a \rightarrow b \rightarrow c) = P(a \rightarrow b)P(b \rightarrow c)$.

2.3 Abstract P2P streaming system

2.3.1 Reduction procedures

The P2P streaming system considered, is an abstract system that generalizes the behaviour of many existing P2P streaming systems of *push* type type, such as Octoshape [13], Lava [14], R2 [15], Splitstream [16] and PPETP. An important characteristic of most of these schemes is that the streams produced by the nodes are not copies of the whole content stream, but are instead *reduced stream* that require a fraction of the bandwidth of the content stream. This approach has several advantages, the most obvious being the fact that in this way, even nodes with small upload bandwidth can contribute to data propagation. Other advantages include a greater reliability of the system, and protection from some attacks such as the *stream poisoning attack* [40], [49]. The details of how the reduced streams are produced are not important for the scope of this work. In this study, the abstract reduction procedure is described by supposing that the P2P streaming scheme defines a set of *reduction functions*[1] $\{\phi_\mu\}_{\mu\in\mathcal{M}}$, indexed by a set $\mathcal{M}$ of *reduction parameters*. Each node selects a reduction parameter $\mu \in \mathcal{M}$ and process each content packet c with the correspondent reduction function to obtain the *reduced versions* $r_\mu = \phi_\mu(c)$ that are forwarded to the lower-peers. The size of r_μ is, typically, a fraction of the size of c. In this study, it is assumed, for the sake of concreteness, that the size of r_μ is R times smaller with respect to the size of c. In this context the details of the definition of ϕ_μ are not interesting, it suffices that the set $\{\phi_\mu\}_{\mu\in\mathcal{M}}$ satisfies the following *R-reconstruction* hypothesis.

Hypothesis 2.1. *(R-reconstruction). Content packet c can be recovered from the knowledge of any set of R different reduced versions $r_{\mu_1} = \phi_{\mu_1}(c), \ldots, r_{\mu_R} = \phi_{\mu_R}(c)$.*

Remark 2.1. *The easiest way to satisfy the R-reconstruction property is by using Reed-Solomon codes [13], [25], [16] but other solutions are possible [50].*

2.3.2 Node behaviour

Supposing that the R-reconstruction hypothesis holds, the typical node behaviour in this model is the following:

1. When a node starts, it choose a reduction parameter $\mu \in \mathcal{M}$ and then, contacts $N \geq R$ upper peers.
2. As soon as the node receives at least R different reduced version of c, it recovers c and moves it to the application level. The node then processes c with ϕ_μ and forwards the results to its lower-peers.

Remark 2.2.

1. *Note that each node* does not *forward the received reduced packets, but it* regenerates *c before creating a new reduced version which is sent to the lower peers.*
2. *If $N > R$ the node receives a redundant set of data. This can be exploited to make the system more robust with respect to packet loss,* churn *and poisoning attacks [40].*
3. *The reduction parameters chosen by the N upper peers must be different from one another. This can be granted by assigning them in a centralized way, but if $\mathcal{M}$ is large enough, peers can choose their parameter at random and still have a small probability of duplicated parameters [40].*

[1] Function ϕ_μ is typically a linear combination (in a finite field) of the data in c but this is not necessary.

4. *The abstract system described here can also be adapted to other schemes such as the round-robin scheme in [31]. To this end, one can consider c as a "macro packet" collecting τ consecutive packets, and defining τ reduction function $\phi_1, \ldots, \phi_\tau$, where ϕ_μ "extracts" the μ-th packet from the macro-packet c.*

2.3.3 Fragment Propagation

It is possible that, because of packet losses, the node receives less than R reduced version of c. In this case the node cannot recover c, but can nevertheless help to continue propagating the information about c, by forwarding to its lower peers one of the reduced packets received by its upper peers. If this happens, it is said that the P2P scheme employs *fragment propagation*.

2.4 Network model

The P2P network will be represented by a Direct Acyclic Graph (DAG) where the edges link each node to its lower-peers. The server(s) will be clearly represented by node(s) that do not have upper-peers. If more than one server is present, it is supposed that they are organized as a CDN and feed their lower-peers with the reduced versions of the same content packet. For the sake of notational simplicity, it is also assumed that every node has $N \geq R$ upper peers and that every link is an erasure channel that drops packets with probability P_ℓ.

Each node m of the network is associated the random variable W_m, defined by the following experiments: the server sends a single packet to the network and $W_m \in \{0, 1, \ldots, N\}$ is the number of reduced packets received by the node m. From the knowledge of the statistical properties of W_m, it is possible to determine several values of interest. For example, the packet loss probability P_{eq} seen by the application can be calculated as $P_{eq} = P[W_m < R]$. As explained in Section 2.3.3, a node sends reduced packets to its lower-peers if it receives at least T reduced packets, where $T = 1$ if fragment propagation is employed and $T \geq R$ otherwise. If node n receives at least T reduced packets (i.e., if $W_n \geq T$) it is said that the node is *active* or in *firing state*. It is possible to define the random variable F_n to be equal to 1 if the node n is in firing state and 0 otherwise.

2.4.1 Limited spread network

One difficulty in studying the behaviour of the abstract P2P system considered here is that the statistical properties of W_n depend on the network topology, a characteristic that is not easily captured by a small set of parameters. In order to simplify the study, it is convenient to put some constraints on the topology.

A useful constraint that is nevertheless general enough to describe practical networks is the hypothesis of *limited spread*. Let n be a node of the network, consider the lengths of the paths joining n with the server and define $d(n)$ and $D(n) \geq d(n)$ as the minimum and the maximum of these lengths. Value $D(n)$ known as the *depth* of node n, and difference $D(n) - d(n)$ is called *spread* of n. The network is said to have Δ-*limited spread* if $D(n) - d(n) \leq \Delta$ for every node n.

The hypothesis of limited spread is quite natural, and it is expected that these types of networks will be the natural outcome of the tentative of maximizing locality. Moreover, it is possible to show that a large spread can increase the jitter experienced by the node, and this suggests that practical networks will keep the spread value as small as possible.

2.4.2 Stratified networks

In this study a special case of limited networks is considered, namely, *stratified* networks[2]. In a stratified network, the nodes can be partitioned into set (*strata*) $S_K, K \in \mathbb{N}$, such that all the upper-peers of a node in S_K belong to S_{K-1}. It is easy to verify that a network is stratified if and only if it has a 0-limited spread and that the stratum index coincides with the node depth. Fig. 2.1 shows few examples of stratified networks, namely a tree network, a network made of parallel trees and a "onion skin" network[3].

2.4.2.1 Notation for stratified networks

Here it is denoted as L_K the number of nodes in each layer represented by K. The n-th node in stratum $K, n = 0, \ldots, L_K - 1$, is named (K, n). The set of upper-peers of (K, n) is represented by the vector $\mathbf{u}_{K,n} \in \{0,1\}^{L_{K-1}}$ whose m-th component is 1 if $(K-1, m)$ is an upper-peer of (K, n) and zero otherwise.

All the random variables $W_{K,n}$ and $F_{K,n}$, relative to the nodes of the stratum K are collected in the vectors $\mathbf{W}_K$ and $\mathbf{F}_K$ defined as

$$[\mathbf{W}_k]_n = W_{K,n} \qquad [\mathbf{F}_k]_n = F_{K,n} \tag{2.1}$$

It is useful to have a special notation for some states in $\{0,1\}^{L_K}$. More precisely, the *empty state* is defined as $\phi = [0, 0, \ldots, 0]$ (no node in active state), the *full state* as $\Omega = [1, 1, \ldots, 1]$ and for every $k \in \{0, \ldots, L_K - 1\}$, the *k-th singleton state*, $\mathbf{e}_k$ as $[\mathbf{e}_k]_n = \delta_{k,n}$ (only the k-th node is active).

2.4.2.2 Constant geometry networks

A stratified network is said to be a *constant geometry* network if every stratum has the same number of nodes (denoted as L in the following) and $\mathbf{u}_{K,n}$ depends only on n, that is , $\mathbf{u}_{K,n} = \mathbf{u}_{M,n}$ for every $M, K \in \mathbb{N}$ and $n \in \{0, \ldots, L-1\}$.

2.4.2.3 Modular networks

A constant geometry network is said to be a *modular network* if the following holds

$$[\mathbf{u}_{K,n}]_m = \begin{cases} 1 & \text{if } m = (n+k) \mod L, k = 0, \ldots, N-1 \\ 0 & \text{else} \end{cases}$$

2.5 Asymptotic Analysis

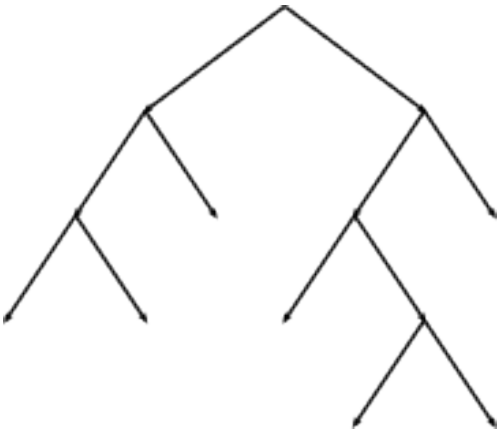
For the sake of notational simplicity, in this section it is assumed that the number of nodes per stratum is constant, and drops the subscript from L_K.

2.5.1 Reduction to the analysis of $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$

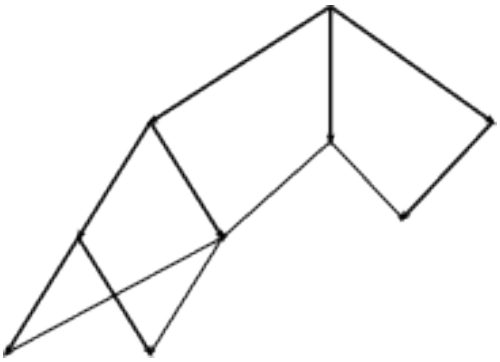
As anticipated, this study is focued on the asymptotic behaviour of the variables $W_{K,n}$ when K goes to infinity. It is clear that, in a stratified network, the random variable sequence $\{\mathbf{W}_K\}_{K\in\mathbb{N}}$ is a Markov

[2] It is used the term *stratified* to avoid confusion with the term *layered* possibly used in other contexts.

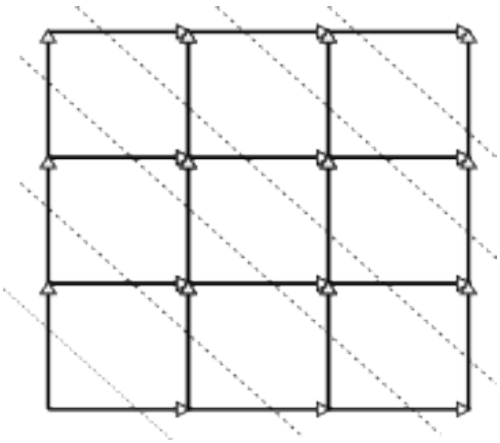
[3] Onion skin networks are interesting because the ratio of non-streaming nodes goes to zero when the network size goes to infinity.



(a)



(b)



(c)

Figure 2.1: Examples of stratified networks (a) tree, (b) parallel trees and (c) onion skin. The dashed lines mark stratum boundaries .

chain with the alphabet $\{0, 1, \ldots, N\}^L$ and that the chain is homogeneous if the network has *constant geometry*. It is immediate to verify that the transition probability is [4]

$$P[W_{K,n} = a | \mathbf{F}_K = \mathbf{s}] = P[\mathcal{B}(\mathbf{s}^t \mathbf{u}_{K,n}, 1 - P_\ell) = a] \tag{2.2}$$

[4]Note that $\mathbf{S}^t \mathbf{u}_{K,n}$ is equal to the number of upper peers of (K, n) in active state.

where $\mathcal{B}(M,p)$ is a binomial random variable with M trials and success probability $1-P_\ell$. According to Eq. (2.2), the transition probability between two different states in $\{0,1,\ldots,N\}^L$ depends only on the pattern of active nodes at stratum $K-1$ and not on the actual number of received packets. By exploiting Eq. (2.2), it is possible to show that the sequence of random vectors $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$ is also a Markov chain. Note that, because of Eq. (2.2), it is sufficient to study the statistical behaviour of the chain $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$.

Let $\mathbf{M}$ be the matrix of transition probabilities

$$\mathbf{M}_{r,c} := P(r \to c) = P[\mathbf{F}_{K-1} = r] \tag{2.3}$$

and let λ_i be the eigenvalues of $\mathbf{M}$, ordered by decreasing modulus, that is, $|\lambda_1| \geq |\lambda_2| \geq \ldots$.

2.5.2 Asymptotic behaviour of $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$

The first, maybe obvious, but important result is the following:

Property 2.1. *The steady state probability of $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$ is*

$$\lim_{K\to\infty} P[\mathbf{W}_K = \mathbf{s}] = \delta_{\mathbf{s},\phi} \tag{2.4}$$

This means that the state of $\{\mathbf{F}_K\}_{K\in\mathbb{N}}$ will eventually converge to the empty state.

Property 2.1 is a consequence of the fact that ϕ is *absorbing*.

Eq. (2.4) could be taken as bad news for the streaming over P2P networks, since it claims that nodes which are "very far" from the server will not receive any packets. In order to first determine what "very far" means, it is important to study λ_2 which, as is well known, controls the velocity of convergence of Eq. (2.4). In order to state the main result about λ_2, we need a generalization of the concept of absorbing state.

Definition 2.1. *Consider a Markov chain with alphabet A and absorbing state $\phi \in A$. A state $s \in A$ is said to be a trapdoor state if there is an integer $\mathcal{L}_s$ such that $P(s \to^{\mathcal{L}_s} \phi) = 1$.*

Note that an absorbing state is a trapdoor, and that a trapdoor state can transitate only to another trapdoor state.

The following theorem gives upper and lower bounds on λ_2.

Theorem 2.1. *Consider the case of a constant geometry network with transition matrix $\mathbf{M}$. Let $\mathcal{T}$ be the set of trapdoor states. If the network is such that Ω can be reached by every non-trapdoor state (that is, $\mathbf{a} \to^* \Omega$ for every $\boldsymbol{a} \notin \mathcal{T}$) then λ_2 is real and strictly larger then $|\lambda_3|$ and the following bounds hold*

$$P(\Omega \to \Omega) \leq \lambda_2 \leq 1 - P(\Omega \to \phi) \tag{2.5}$$

Proof. In this demonstration, some simple details are skipped. Transition matrix $\mathbf{M}$ can be written in the form:

$$\mathbf{M} = \left[\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline * & \mathbf{H} & 0 \\ \hline * & * & \mathbf{Q} \end{array}\right] \tag{2.6}$$

where the first row (and column) corresponds to the absorbing state ϕ, the second block of rows (and columns) correspond to the trapdoor states and the third block to non-trapdoor states.

It is clear that $\lambda_1 = 1$ and the remaining eigenvalues of $\mathbf{M}$ are distributed between the eigenvalues of $\mathbf{Q}$ and the eigenvalues of $\mathbf{H}$. By definition of the trapdoor state, it follows that $\mathbf{H}$ is nil-potent and that all the non-zero eigenvalues of $\mathbf{M}$ are eigenvalues of $\mathbf{Q}$.

In order to prove that the λ_2 is strictly dominant, one shows that $\mathbf{Q}$ is irreducible and primitive by observing that (i) for every $\mathbf{a}, \mathbf{b} \notin \mathcal{T}$ one has $\mathbf{a} \rightarrow^* \Omega \rightarrow \mathbf{b}$, and (ii)$P(\Omega \rightarrow \Omega) \neq 0$.

The lower bound in Eq. (2.5) is a result of the fact that the spectral radius of a non-negative matrix is never smaller then the diagonal elements. The upper bound in Eq. (2.5) is a result of the fact that in a non-negative matrix the spectral radius is not larger than the maximum row sum, that is

$$\lambda_2 \leq \max_{r \in \mathcal{T}^c} \sum_{c \in \mathcal{T}^c} P(r \rightarrow c) = \max_{r \in \mathcal{T}} 1 - P(r \rightarrow \mathcal{T}) \tag{2.7}$$

Since ϕ is a trapdoor state, $P(r \rightarrow \mathcal{T}) \geq P(r \rightarrow \phi)$ one can infer that

$$\lambda_2 \leq \max_{r \in \mathcal{T}} 1 - P(r \rightarrow \phi) = 1 - P(\Omega \rightarrow \phi) \tag{2.8}$$

where the last equality results from the fact that $P(r \rightarrow \phi)$ is minimized when $r = \Omega$.

□

It is worth explicitly writing the probabilities in Eq. (2.5)

$$P(\Omega \rightarrow \Omega) = P[\mathcal{B}(N, 1 - P_\ell) \geq T]^L \tag{2.9a}$$

$$P(\Omega \rightarrow \phi) = P[\mathcal{B}(N, 1 - P_\ell) < T]^L \tag{2.9b}$$

that in the case of fragment propagation become

$$P(\Omega \rightarrow \Omega) = (1 - P_\ell^N)^L \approx 1 - L P_\ell^N \tag{2.10a}$$

$$P(\Omega \rightarrow \phi) = P_\ell^{NL} \tag{2.10b}$$

Exemple A simple numerical example can help us to understand the meaning of Theorem 2.1. Consider the case of fragment propagation, $P_\ell = 0.1, N = 8$ upper-peers per node and $L = 100$ nodes per stratum. According to (2.5) and (2.10a), λ_2 is not smaller than

$$P(\Omega \rightarrow \Omega) \approx 1 - 100 \cdot 0.1^8 = 1 - 10^{-6} \tag{2.11}$$

It therefore easy to confirm that in order to have $\lambda_2^K < 0.99$ it is necessary to have $K > 10^4$. This shows us that although nodes that are "very far" away will receive very few packets, the convergence is very slow and it is quite unlikely that one will find "very far" nodes in practical contexts.

In the case of no fragment propagation and $R = 5$, the lower bound λ_2 is

$$P(\Omega \rightarrow \Omega) = P[\mathcal{B}(8, 0.9) \geq 5]^{100} \approx 0.6 \tag{2.12}$$

which is much smaller than (2.11). Although this is only a lower bound, it suggests that convergence to the empty state can occur very quickly, if fragment propagation is not used.

Theorem 2.2. *The reconstruction probability $1 - P_{eq}$ experienced by a node at stratum K can be lower bounded as*

$$1 - P_{eq} \geq P(\Omega \rightarrow \Omega)^{K-1} P[\mathcal{B}(N, 1 - P_\ell) \geq R] \tag{2.13}$$

Theorem 2.2 can be proved by observing that even "the node (K,n) recovers the packet" includes the event "all the strata from *1* to $K - 1$ are in the full state and node (K,n) receives at least R out of the N packets sent by its upper peers". Furthermore, the right hand side of Eq. (2.13) is the probability of the latter event.

Note that, if the fragment propagation is employed, the term $P(\Omega \rightarrow \Omega)^{K-1}$ in (2.13) can be made as close to 1 as desired without increasing the redundancy, since it depends only on $P[\mathcal{B}(N, 1 - P_\ell)] \geq R$. According to (2.13), if $P(\Omega \rightarrow \Omega)$ is large enough, the node in stratum K does not "notice" the presence of $K - 1$ strata between itself and the server, and it experiences a packet loss probability on every link almost equal to the "basic" loss probability P_ℓ.

2.5.3 Extension to more general cases

2.5.3.1 Non-constant geometry stratified networks

The results above have been derived within the hypothesis of a constant geometry network. However, the bounds in Eq. (2.5) do not depend on the connections between consecutive layers and this suggests that a similar result can also hold in the case of non-constant geometry networks.

Moreover, note that bound (2.13) holds *exactly* even in non-constant geometry networks since $P(\Omega \to \Omega)$ does not depend on the connection pattern.

2.5.3.2 Non-stratified networks

If $P(\Omega \to \Omega)$ is large enough, the decay is so slow that, intuitively, it should not make much difference if the node at stratum K receives its data from layer $K-1$ or $K-\Delta$, as long as Δ is not too large. This suggests that, at least in the $P(\Omega \to \Omega) \approx 1$ case, similar results could also hold for non stratified networks.

2.6 Simulation results

Some simulations were carried out in order to complement the analytical results. To obtain the following results, three different types of networks have been utilized:

- A *modular network*.
- A *constant random network*, which is a constant geometry network where vectors $\mathbf{u}_{1,1}, \ldots, \mathbf{u}_{1,L}$ are independently drawn from a set U_N of vectors in $\{0,1\}^L$ with N entries equal to one.
- A *totally random networks* where every $\mathbf{u}_{K,j}$, is a randomly drown from U_N.

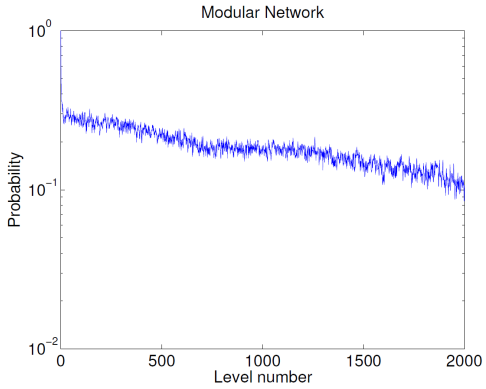
For every network, packet propagation across strata has been simulated 100 times, when the frequency of packet reception at each node was measured. In the case of random networks (i.e., constant random and totally random networks) the above procedures were repeated 20 times, with randomly chosen link layouts, and the results were averaged.

Figs. 2.2(a),(b) and (c) show the measured probability of receiving at least one packet for, respectively, the modular network case, the constant random network case and the totally random case. In every case, the following were utilized $L = 10, N = 3$ and $P_\ell = 0.5$.

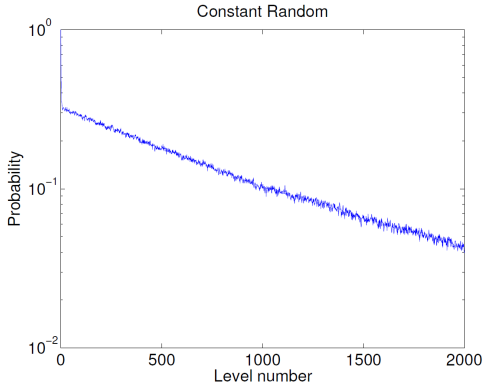
Remark 2.3. *Such a large value of P_ℓ was chosen in order to have the probability decay visible. Even slightly smaller values of P_ℓ make the decay almost unnoticeable.*

Note the exponential decay of the probability in the constant geometry networks (Fig. 2.2(a) and Fig. 2.2(b)) as predicted by the theory above. However, note as well, that the same exponential decay happens in the case of Fig. 2.2(c), corresponding to a non-constant geometry network, supporting the claim in Section 2.5.3.1.

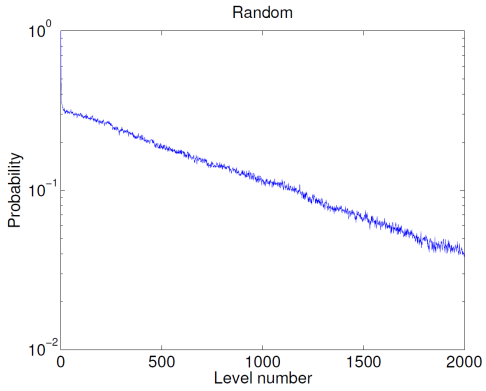
Note that the two random networks (Fig. 2.2(b) and Fig. 2.2(c)) exhibit a very similar decay rate, faster than the decay rate of the modular network (Fig. 2.2(a)). Although more investigation is necessary to fully understand this phenomenon, it is supposed that this is due to the fact that, in a modular network, every node has exactly N lower-peers, while in a random network the number of lower-peers may vary (even if the number of upper-peers of each node is fixed to N). In a random network if a node at stratum K does not receive any packet and has more then N lower-peers, many lower-peers at stratum $K+1$ will experience a higher loss probability, thereby possibly making the network more fragile.



(a) Modular Network



(b) Constant Random



(c) Random

Figure 2.2: Probability of packet reception in (a) a *modular* network, (b) a *contant random* network and (c) a *totally random* network. In all the plots $L = 10, N = 3$, and $P_\ell = 0.5$.

2.7 Conclusions

This study analyzed the packet loss probability P_{eq} experienced by the application when a stream-based, *chunkless* network is employed. It was shown that, although in the limit P_{eq} converges to 1, it is possible choose the network parameters to make this convergence so slow that such probability remains negligible in networks of practical size. Moreover, such a convergence can be made as slow as desired without increasing the redundancy in the network.

Chapter 3

PPETP: Peer-To-Peer Epi-Transport Protocol

3.1 Introduction

PPETP is a protocol initially developed for the video streaming over P2P networks on which the nodes have enough download bandwidth to receive the entire stream but not enough upload bandwidth for its retransmission. Today this is a very common condition, considering that a good quality video, obtained with a high performance codec as H.264/AVC needs at least a bandwidth of 300 kbit/s[1] and the traditional residential lines, working with the Asymmetric Digital Subscriber Line (ADSL) technology usually provide some Mb/s of download bandwidth and few hundred kb/s of upload bandwidth. To efficiently exploit the bandwidth characteristic of each user, PPETP uses mainly NC, which allows a reduction in the bandwidth requested, in order to upload the stream without decreasing its quality.

The structure of PPETP has evolved in past few years, and has now become a sophisticated and very versatile protocol that could be considered as a multicast overlay protocol based on a P2P structure, also able to efficiently exploit the nodes with a limited upload bandwidth. With this protocol, the transport of all kind of information is possible, not only of multimedia, which makes this protocol adaptable to many different applications. As will become clear later in this thesis, applications view PPETP as simply a transport protocol, similar to the IP multicast. A task of the protocol is the management of the peers, and of the messages necessary for a correct exchange of information among them. Differently from many others P2P solutions, PPETP does not impose a specific network structure or bindings in the nodes selection, but focuses uniquely on the transport level. The network characteristics are chosen by the applications, and in this way the best solution can be applied for the context. For example, in same circumstances, a centralized system can be used to select the peers that a node must contact. This allows for accurate control of the entire system. In others contexts a distributed method for the peers selection, for example, with a Distributed Hash Table (DHT) system can be chosen. At first glance this could seem to be a deficit of the protocol, but on the contrary, this allows for an increasing in versatility. Other transport protocols do the same, for example in TCP specifications it is not specified as the IP address and the ports must be determinate, it supposes that these are known.

In the following chapters, the latest developments in PPETP are described, which is under devel-

[1] For HD video the stream arrives at some Mb/s.

opment at the University of Udine within the Sourceforge project *Corallo* and the Ministry founded program PRIN[2] *Arachne*.

In order to better understand the building details of PPETP, an introduction and short overview of its main functionalities is provided. In this way, it is possible to become familiar with most of the aspects of the protocol without focusing too much on the implementation details. After this deliberately short overview, some typical applications are presented that can find benefit with the use of a P2P network, and in particular by PPETP, because its characteristics are very well adapted to solving these problems, without discharge the efforts of the network management on the applications.

After these two sections, a the detailed presentation of PPETP will begin. It was chosen to not include every single particular of the protocol in the thesis, but only enough details to estabilish a proper undestanding of the protocol's behaviour. When necessary, it is possible refer to the Internet Engineering Task Force (IETF) draft [25] for more details. PPETP is still under development, and the information contained in this thesis are up to date with the actual, quite stable, version of the protocol (December 2011). However for for more up to date information it is better to refer to the site version of the draft.

3.2 Overview

As mentioned above, PPETP was born from an application for video streaming over P2P networks, therefore it has inherit a number of features from this application. In this section some key features are introduced in an informal manner, and in this way it is possible to have an overview of the characteristics before addressing with them in a more complete form.

The first important characteristic is the possibility of utilizing NC as a tool to face the problem of the band asymmetry. The operations of the NC, or as we will see later, the operations of encoding and decoding in general, take the name of *Reduction profiles* and include a complex series of operations that are addressed in Section 3.6. Normally, a peer is connected to more lower-peers, and in this case it is natural to give the possibility give different coding parameters to different lower-peers (in this way it is possible to produce different versions of the output contents). This characteristic is obtained by means of *Channels* (see Section 3.7) that are sets of peers sharing the same coding parameters.

Another aspect very important in the real P2P applications is "peers reachability". Often in the P2P papers the operations needed because the nodes are able to communicate each other are omitted, but they merely explain the algorithm on a conceptual level rather than in a real one. This often suggests, to non-experts, that this is always an easy operation, in fact it is not the case. Often, the users (particularly the residential ones) are not directly connected to the network, but are connected to it by means of firewall or NAT [18], as shown in Fig. 3.1.

These devices normally make it very difficult, if not impossible, to contact these users from others users that are on the opposite side of these devices. To resolve this problem, but not in all the cases, some techniques called *Connection Establishment Procedures (CEP)* are available. These are procedures that permit the hosts to exchange their addresses list with each others, and the selection of a pair of addresses means the hosts should be able to communicate. In the Section 4.4 is explained briefly a CEP called Interactive Connectivity Establishment (ICE).

The use of a classic IP address and a port makes no sense in these contexts because they are not sufficient to grant a connection in all the conditions. To effectively identify the peers, also in the pres-

[2]PRIN: Research Project of National Interest.

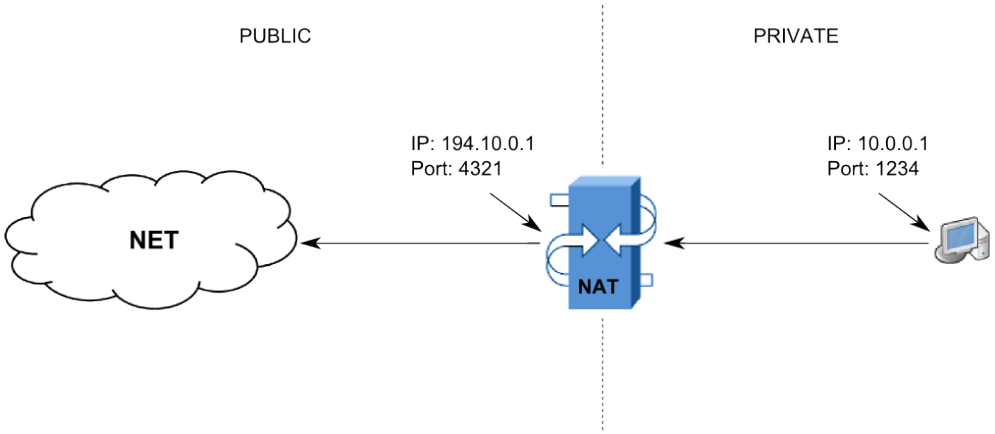


Figure 3.1: Connection of a computer to Internet by mean of a NAT.

ence of the NATs, the *generalized address* (Section 3.5) is utilized, which contains all the information necessary to contact the peers. In case a peer is not covered by a NAT, its generalized address coincides with the IP address and port.

A P2P application is a distributed system in which a large number of users exchange data each other. There are two kinds of packets: the first type are the *content data* (Section 3.8.1) which content is the applications payload, and the latter consists of *commands* (Section 3.8.2) that application exchange each other for the management of the entire system.

Because of these exchanges of information between users, it is also possible to have security problems that must be considered, both for commands and contents information. It is possible, in fact, that users deliberately inject incorrect data, with the risk that the entire system suffers the consequences. There are many different attacks, some of them are reported in Section 4.1, which the protocol (or in some cases the applications that use it) should remedy. PPETP gives some features that grant that the packets received effectively come from the desired users, and the information that they contain are correct and have not been modified. This feature is obtained through appropriate authentication functions based on cryptology (Section 4.1).

Because of timing problems, stream-oriented applications usually use UDP packets for the transmission of the data. PPETP also can use these kind of packets. The use of this protocol is preferable because, differently for example from TCP, there is no retransmission of lost packets and congestion control, which slow down the stream transmission. This is not acceptable in real time applications, because a packet that arrives too late is useless for the video decoder. On the other hand, UDP does not have a congestion control, which means that in presence of congested links, the source continues to send packets at the same rate, increasing the congestion. For this reason, the IETF suggests the utilization of a control rate algorithm [51]. PPETP accepts this advice and provides the TCP Friendly Rate Control (TFRC) algorithm [52] for controlling the rate (Section 3.9). In any case PPETP is compatible with both TCP and Datagram Congestion Control Protocol (DCCP) [53] that provide rate control. The latter protocol is similar to UDP, but natively provides a congestion control algorithm more “stream friendly” respect TCP. Unfortunately, DCCP is not widely utilized because some operating systems do not implement it, for this PPETP is used as default UDP with an implementation of the control rate rather than leave this purpose at the underlying transport level.

In many circumstances the stream can be composed of many different "sub-streams" with different characteristics. Typical cases are the streaming services (Section 3.3.1), where a single server can distribute different streams for audio, video and text content, and also in conferencing (Section 3.3.3) and gaming (Section 3.3.4) environments where there are many sources that "share" the same stream. In all these cases it is necessary to distinguish between the different streams. With PPETP this is obtained by introducing the concept of the *stream_ID*, which is simply an identifier associated with the different "sub-stream".

In some contexts it can be useful to assign different levels of importance to the packets. A typical example of this can be when a LC codec is utilized. As seen in the introduction, the layers have different importance; the base layer is the most important and the following enhancement layers have decreasing importance. PPETP manages the "importance" of the packets allowing us to assign them a *priority class*, that is an 8-bit integer. It is not specified how the classes must be assigned, the only constraint is that the priority must be a non-decreasing function of the value of this field meaning that class 0 has the highest priority.

The priority class is used in the following context:

- It may be utilized in the reduction procedures (Section 3.6) to adapt the reduction parameters in function of the class.
- In the congestion control procedures (Section 3.9), to drop packets in function of their class priority.
- It can be used in the puncturing procedures (Section 3.10), associating different lose probability to different classes.

3.3 Typical Applications

There are a number of applications that have the necessity to share data with a large number of users in a multicast fashion. The most known applications for this kind of data transport are surely the streaming (live or not) of video and audio contents, software update, conferencing and gaming, but other applications also can be developed. All of these applications can use a P2P approach to deliver data to their clients in a more efficient and economic manner.

3.3.1 Streaming

Probably, the most common use of a protocol such as PPETP is the streaming of video and/or audio data. Very often, these kinds of multimedia contents are intended for a large number of users, spread over the network. Considering the large requirement of bandwidth for these streams, a multicast delivering is optimal to decrease the costs for the content and network providers and to avoid network congestions. It is necessary to place the stream applications into two mains categories: *Live Streaming* and *On-Demand Streaming*.

The first category is comparable to the traditional radio and television systems where there is a content broadcaster that transmits a program on the air. A user can be tuned and receive the transmission, can turn off the receiver and turn on again: what is happened in the middle is lost and what will happen in the future is yet unknown, therefore it is said that the transmission is *real time*.

The latter category, the on-demand, is comparable with the use of a CD or DVD player: the user can play the media when they want, it can pause it, re-play, etc.

While using traditional radio and television is easy to realize real time systems and is more complicated to develop the on-demand ones, working in the Internet environment the opposite is true. With the Internet, the on-demand streaming is simply a particular kind of file download from a server, therefore it is sufficient that the downloaded data allows the reconstruction of the media with respect to the play-time scheduling. However, if the user has enough bandwidth it can also download the following data because the entire file is already present in the server. In live streaming, this is not possible because a file does not exist on the server. The media is produced and immediately sent to the network, and, for correct decoding the data must arrive at the receiver before their play-time. Otherwise, if the data arrives too late, it is useless, and the decoder is unable to reconstruct the correct media. It is possible, and usually is done, to bufferize some amount of data in the user's application to have a little delay margin, but this degrades the quality of the system.[3]

When both these systems have a small number of users, the use of a unicast stream is perhaps the best solution because of its simplicity, but when this number grows this solution cannot be adopted. As said in the introduction the best solution is the multicast IP. Because this architecture is infeasible on a large scale, another approach is necessary, as was proposed with PPETP which is a multicast overlay, based on a P2P network. Every peer that is downloading a stream provides, compatibly with its upload bandwidth, to retransmit it toward other users. With this approach the server, and also partially the network[4] is unload from the huge traffic requested by the total download bandwidth. Also for the on-demand content, this approach is possible, with an accurate project of the applications that use PPETP. In this case the user that wants to download a stream contacts some users that have already downloaded it. In fact, if these users have saved a little cache of the downloaded file on their disk, they are able to retransmit it toward the new user.

3.3.2 Software update

Nowadays, lots of software such as operating systems and antivirus software require periodic, if not daily, updates. Also in this case, as for streaming, a lot of users update their software simultaneously or in a short amount of time, hence the servers responsible for the update are heavily loaded. A very simple solution is that these servers work like broadcaster repeatedly sending their data as a stream over PPETP. In this way, the software that must be updated is able to receive an entire update by staying connected to the "broadcasting" for a sufficient amount of time.

3.3.3 Conferencing

Thanks to new technology that allows this kind of communication, tele-conferencing is one of the most desired application for use mainly by companies, but also by individuals. Many companies have developed products in this sector[5], but they are not compatible with each other. Therefore, all the clients must use the same system, limiting a large diffusion of these systems. Considering that normally, a company interacts with many other companies, and a common protocol shared by different products, would probably would allow for a larger diffusion of these systems. Differently from the two previous applications where there was a single source and many users, in conferencing applications every user is also a source. The communication delay is also very important for this application because the users interact each other, and a large delay[6] makes the interaction very difficult. A trivial solutions to this is

[3] For example in a multi-channel system, when the user changes channel, it must wait until that the buffer is full enough before the media is played.

[4] It depend also from the P2P network structure.

[5] For example HP with *Halo*, Cisco with its *Cisco TelePresece Systems* and many others.

[6] Often greater than some hundred milliseconds.

the use of a central server that receives the stream by all the participants at the conference, mixes the streams, and re-sends a new stream to all the users. In practice, this scheme can work only for a small number of users, beacuse of both server bandwidth and processing power. PPETP manages this scenario in a very interesting manner, by allowing the streaming of one or more flows produced by one or more sources indicated simply by a stream identifier.

3.3.4 Gaming

More than conferencing, nowadays network games are very popular. In these games, all users share the same game's environment, collaborating to reach a common objective or challenge to win a match. The scenario is very similar to that of conferencing applications, as there are many users that are also the source of the data. Very often, the time requisition are also as restrictive as with conferencing. PPETP can also be used to reach the objective of exchanging data among all the participants.

3.4 Characteristics of PPETP

Following this general overview of PPETP, as covered in the previous section, the most important details are presented here. These characteristics are almost sufficient to make the protocol work, however, some other important details are included in the next chapter.

3.5 Generalized Address

To contact a host in the Internet, its IP address and the port on which the application is listening must be known. Nowadays, most of the residential users are behind NATs, which means that having only the IP and port is not sufficient to contact them, because only the applications know the internal IP and port of the host, while the outside world knows only the public IP address and port of the NAT as depicted in Fig. 3.2. Moreover, the NATs imposes some roles denying packets from the outside to pass through the

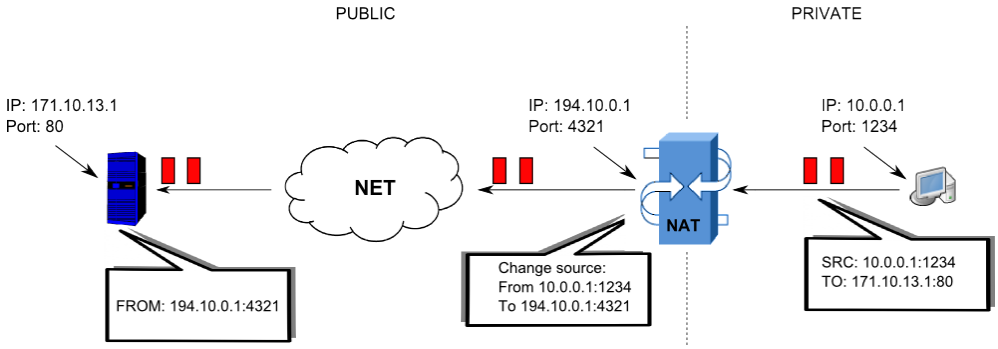


Figure 3.2: Change of address due to the NAT.

NAT if not authorized by a role. The same thing applies if a firewall is present. In order to contact a host behind a NAT, some CEP exist that in some cases allow the host to be contacted. An example of these procedures is the ICE explained in [54], which is the default CEP of PPETP (Section 4.4). To identify a PPETP host, the *generalized address* is used, which is a used to describe IPv4 and IPv6 addresses, or to describe the data necessary for the CEP. Currently, there are two classes of addresses:

IP This class contains an IP address (IPv4 or IPv6) and a port. It is used when a host is directly connected to the network by means of a public address.

ICE This class contain information for the CEP based on ICE protocol. It contain the address of the bridge server and an ID of the remote host to contact.

3.5.1 Generalized addresses structure

When it is necessary to include the generalized address in a packet it must be converted into a binary format, whose structure is reported in Fig. 3.3. The five most significant bits of the first byte represent

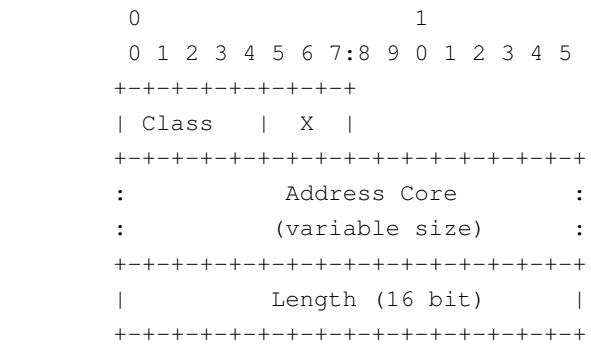


Figure 3.3: Generalized address structure.

the address class (for a total of 32 possible classes) while the following three bits depend on the class. Actually, class 0 is defined as that is the generalized IP address, and the class 1 is defined as the ICE class. Classes from 2 to 30 are undefined, and class 31 is reserved for future extensions. The last 2 octets is an integer number representing the total length of the structure. The length is at the end of the structure because in same context (like routed packets described in Section 3.8.3) it is easier start reading from the end of the structure. The remaining bytes are the address data, whose format depends on the class.

3.5.2 IP address class

The format for the IP address class is reported in Fig. 3.4, and the meaning of the fields are the following:

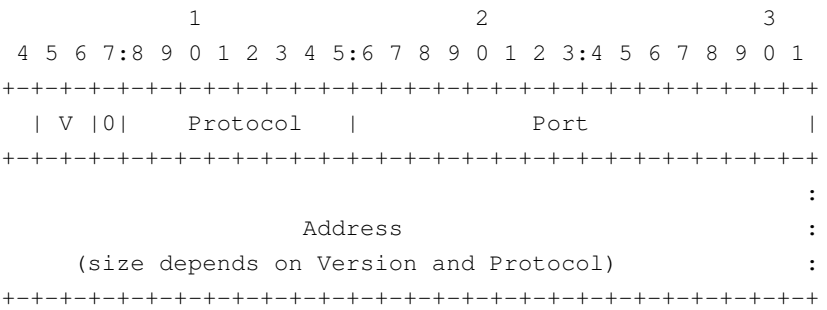


Figure 3.4: IP address core format.

address class (bits 0-4) This field indicates the IP class address, and it is set to 0.

version (bits 5-6) Indicates the version of the IP protocol: if its value is 0 it indicates an IPv4 address. Otherwise if it is 1, it indicates an IPv6 address. The other two values are reserved for future extension.

unused (bit 7) This bit is not used, and must be setted to 0.

port (bits 16-31) If the transport protocol is based on ports (e.g., UDP, TCP and DCCP) this field indicates the port of the destination host, otherwise it is set to 0.

address Contains the address of the host, and the size of this field depends on the version field V. For an IPv4 address, its length is 32 bits, while for an IPv6 address its length is 128 bits.

3.5.3 ICE address class

The format for the ICE address class is reported in Fig. 3.5, and the meaning of the fields are the following:

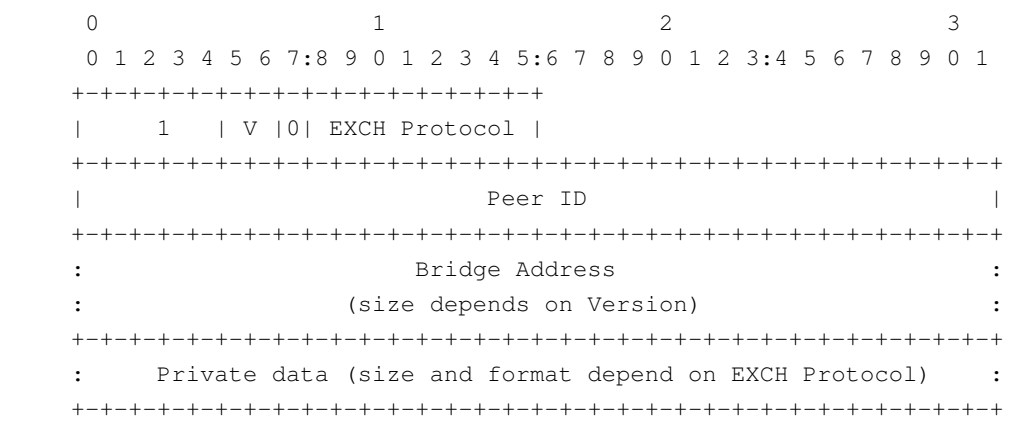


Figure 3.5: ICE address core format.

address class (bits 0-4) This field indicates the ICE class address and it is set to 1.

version (bits 5-6) Indicates the version of the IP protocol of the bridge address: if its value is 0, it indicates an IPv4 address. Otherwise if it is 1, it indicates an IPv6 address. The other two values are reserved for future extensions.

unused (bit 7) This bit is not used, and must be set to 0.

EXCH Protocol (bits 8-15) Indicates the procedures used by the nodes to exchange the ICE candidate, here the values can assume: 0 if an HTTP-based procedure is used, 1 that is equal to the protocol 0, but it uses HTTPS. The values from 2 to 254 are unused and the value 255 is reserved for future uses.

Peer ID The 32 bit PEER_ID of the remote host.

Bridge address The IPv4 or IPv6 address of the bridge node.

Other data This field can be used to store data that can be used by the candidate exchange protocol. The size and the format of this field depends on the field EXCH PROTOCOL.

3.6 Reduction profiles

In PPETP, there is a deep separation between the management of all the aspects of the protocol and treatment of the data. A first distinction is at the packets level, where there is a separation between *control packets* and *data packets*. The firsts are elaborated, as explained in Section 3.8.2, by the "core" of the protocol, while the latter are elaborated by mean of *Reduction Profiles*. The reduction profiles

are a set of procedures (parameterized) that define how to elaborate the data for the *reduction* and the *reconstruction*. The term reduction[7] is used when the data are elaborated for the transmission, and *reconstruction* when the data are elaborated after the reception. Currently two different reduction profiles are defined: the *Basic* profile that does not make any operation in either reduction or reconstruction, and the *Vandermonde* profile that is a NC with a particular structure of the reduction matrix, as reported in Section 3.6.1. The basic profile was introduced in PPETP for its simplicity. It does not have an algorithm for reducing the size of the data, or to change its format, hence it was developed mainly for test purposes or for very low bandwidth stream, while the Vandermonde profile is the main profile normally used in the transmission of data. It is easy to add other reduction profiles, since the data packets are not aware of the content of their payload, seen simply as a byte sequence. It is only in the configuration phase that the used profile is specified. This choice allows us to not limit the protocol to a little set of reduction profiles, but allows us to expand in the future or to specific features. Normally, the reduction function produces only one reduced packet for every data packet, but this is not mandatory. The reconstruction procedure generally takes place after the reception of an adequate number of reduced packets, enough to sufficiently reconstruct the original data. After the reconstruction, the data is passed to the application and to the channels to be reduced again and sent to the lower-peers.

3.6.1 Vandermonde profile

The Vandermonde profile is the main profile of PPETP, and was the idea on which this protocol was developed. All of the proprieties of the reduction with this profile are described in [40], while the detailed construction of the profile are described in the Appendix A.1.3. In this section the key points of this technique are reported.

In the Introduction, the general concept of NC was presented, and the Vandermonde profile is a particular kind of this technique. Let call R the reduction factor and $\mathbf{P}$ the content packet that must be reduced. $\mathbf{P}$, possibly padded, is organized as a matrix of R rows of elements of the finite field $GF(2^d)$ where R and d are fixed in the configuration phase. R is the reduction factor, while d is used to specify the dimension of the finite field, as explained in Appendix A.1.3, where $d = 8 \cdot gf\_size$, in PPETP this parameter can assume the value of 1,2 or 4. Always in the configuration phase, an element b of the $GF(2^d)$ is chosen, and with this the reduction vector $\mathbf{r}_b = \left[1, b, b^2, \ldots, b^{R-1}\right]$ is constructed. The reduced packet $\mathbf{u}_b$ is obtained computing

$$\mathbf{u}_b = \mathbf{r}_b \cdot \mathbf{P}$$

This operation collapses all the columns of the matrix into single elements, so in this way $\mathbf{u}_b$ is a vector about R times smaller than $\mathbf{P}$.

The reconstruction of the content packet is not feasible through the reception of a single reduced packet $\mathbf{u}_b$, but at least R packets constructed with a different reduction vector $\mathbf{r}_b$ are necessary. This means that of all these vectors have been constructed by different elements b_i. With the reception of the R packets $\{\mathbf{u}_{b_1}, \mathbf{u}_{b_2}, \ldots, \mathbf{u}_{b_R}\}$ it is possible to construct the linear system:

$$\begin{bmatrix} \mathbf{u}_{b_1} \\ \mathbf{u}_{b_2} \\ \vdots \\ \mathbf{u}_{b_R} \end{bmatrix} = \begin{bmatrix} 1 & b_1 & b_1^2 & \cdots & b_1^{R-1} \\ 1 & b_2 & b_2^2 & \cdots & b_2^{R-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & b_R & b_R^2 & \cdots & b_R^{R-1} \end{bmatrix} \cdot \mathbf{P}$$

The matrix with the powers of b_i elements is a Vandermonde matrix and could be inverted when all the coefficients b_i are different. This is a sufficient condition because the determinant of a Vandermonde

[7] It is used the term *reduction* because the size of the data after this procedure is often smaller than the initial size.

square matrix $\mathbf{B}$ of order n and coefficients α_i is

$$\det(B) = \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i) \tag{3.1}$$

which is not zero if all the coefficients are different form each other.

3.7 Channels

Nodes in PPETP can produce more than one version of reduced packets, processing it several times with different parameters[8]. Each different version is said to have been produced by a *Channel*. At every channel an arbitrary number of lower-peer can be connected, limited only by the upload bandwidth, while the maximum number of channels is 16. It should be considered that every reduction operation requires processing power; therefore for highly complex reduction profiles this consideration should be taken into account.

3.8 Packets

The packets exchanged by the peers that use PPETP can be classified as *Data Packets* (Section 3.8.1) and *Control Packets* (Section 3.8.2). Obviously, the first are used to transport the reduced packets, while the latter are used to transport the commands, used in all the operations of stream flow control and for the management of the P2P network. The command packets generally need to be acked while the data packets no (see Section 3.8.2.3). In the next sections, only the format and the main characteristics of the packets are described, while the complete description of all the fields are described in detail in the draft [25].

3.8.1 Data Packets

The data packets are the most frequent ones in a PPETP session, in fact, they are designed to transport all the reduced stream's data. The structure of these packets is depicted in Fig. 3.6.

Without entering in the details of the fields V and C are used to determine the protocol version, and to distinguish data packets from control packets, the flag P indicates if the payload is padded and the flags I,F,G,H are used to give information about the payload. Their meaning depend on the reduction profile, with one singularity: the meaning of flag I, that indicates if the reduction parameters are inserted into the payload (see Section 3.8.1.1). The TIMESTAMP with ERT and RTT fields are parameters used by the congestion control mechanism (see Section 3.9), while the PPETP MAGIC is a constant whose decimal value is 95 and it is used to simplify the distinction between the PPETP packets and the others. This field is necessary because in the same port (port of the transport protocol e.g. UDP) used by PPETP other kind of packets can be received. For example, it is necessary to use this port to receive STUN packets when the ICE protocol is used. The SENDER SIGNATURE is used to authenticate the packet. This signature is applied by the upper-peer who produces the packet and is used to avoid a defamatory attack (Section 4.1). The remaining fields contain information about the stream. The SEQUENCE NUMBER is a sequential number assigned the content packet. The CHANNEL is the PPETP channel that it is used to specify which reduction parameters were used to reduce the content of this packet (see Section 3.7) and the STREAM ID is a stream identifier of the original content packet. The CLASS identifies the priority class of the packet. The value 254 is reserved for future extensions, while the value 255 is invalid.

[8] For example, using different b_i when the Vandermonde profile is utilized.

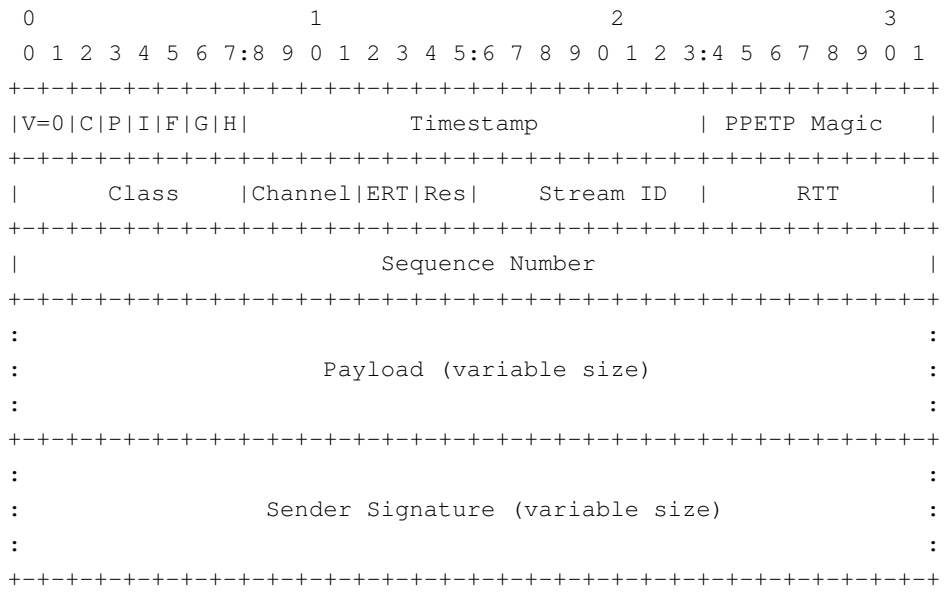


Figure 3.6: PPETP Data Packet.

The last field is the PAYLOAD which obviously contains the data produced by the reduction profile, eventually padded[9].

3.8.1.1 Motivation of the I flag

With NC, a node after the reception of an adequate number of reduced packets reconstructs the content packet and then reduces it again with its parameters. In the case that the node does not receive enough reduced packets, it cannot propagate its own version of the reduced packet to its lower-peers. This decreases the probability that the lower-peers are in turn able to reconstruct their content packet. To remedy this problem a node that is unable to reconstruct the content packet can forward a reduced packet toward its lower-peers that is received by its upper-peers. This operation is also called *Fragment Propagation* as explained in Section 2.3.3. The problem of this method is that lower-peers do not have the reduction parameters used to reduce these packets, for this the node must insert them into the packets before sending them. The I flag is used to indicate the presence of these parameters.

3.8.2 Control Packets

The second kind of packets that are presented in PPETP are the control packets. These types of packets are designed to transport information for the management of the network and for the stream flow control. The behaviour of the peers in the presence of the control packets is different with respect to that of the data packets. Different from the data packets, control packets are not processed by the reduction profiles, but are processed by the "core" of PPETP. After this, an acknowledged packet is sent to confirm the reception and execution of the command (Section 3.8.2.3). The structure of these packets, depicted in Fig. 3.7, is not much different from that of the data packets (Fig. 3.6), but the complexity of these packets resides in the payload. For every kind of request a particular payload containing all the information needed for the execution of the command is defined.

There are a number of fields that have the same meaning as the correspondents of the data packets,

[9]It could be not only the data produced by the reduction function, for example if the flag I is set there are also the reduction parameters.

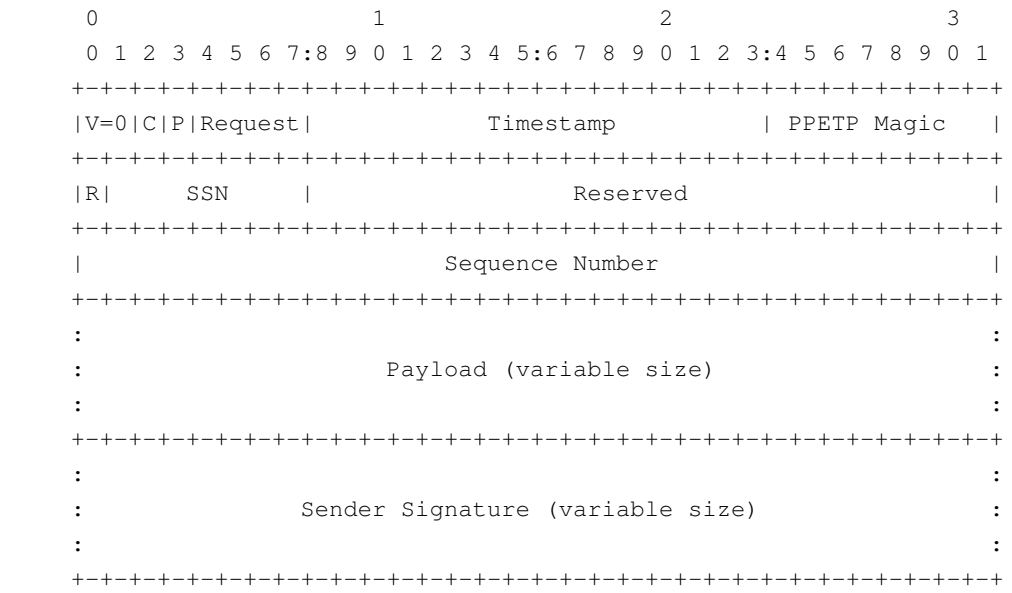


Figure 3.7: PPETP Control Packet.

and these are the flags V, C, P, the fields TIMESTAMP, PPETP MAGIC. The SEQUENCE NUMBER also has the same meaning as that the data packets, but these sequence numbers are not shared with them of the data packets, hence is possible to receive a data packet and a control packet from the same sender with the same sequence number. The last field in common is the SENDER SIGNATURE. The new fields are the fields REQUEST that contain the command type (see Section 3.8.2.1) the flag R that indicates if the packet is a routed one (see Section 3.8.3), the sub sequence number SSN that is used together with the SEQUENCE NUMBER to identify a single control packet (see Section 3.8.2.3). The last field is the PAYLOAD that contains the request information.

3.8.2.1 Request types

There actually are 9 different commands in the PPETP protocol. Some of them (SET_PARAMETER, ACKNOWLEDGE, CLOSING, HELLO, OPEN_CONNECTION and FEEDBACK) are used for generic operations, while the others (START, STOP and REDIRECT) are specialized for the flow control between the nodes. The latter typology of packets could also be sent, for example, by a control entity within the scope of network management. With the exception of the ACK and FEEDBACK, all the requests are required to be acknowledged. The request types are:

Set_Parameter (Request 0): This request is sent from an upper-peer to a new lower-peer during the handshake phase to communicate the set of the reduction parameter used. The payload consists on a set of attributes (see Section 3.11) in the Type Length Value (TLV) format (see Section 3.8.2.2). The payload can also be empty, in this case its meaning is a no-op and it is used to trigger an acknowledge packet which is used to keep a NAT hole open, or to check if a peer is still alive.

Acknowledge (Request 1): This type of control packet is used to acknowledge the receipt of the other control packets. The payload is formed by the sequence number and SSN of the packet to be acknowledged and by an error code (see Tab. 3.1). This request does not require acknowledgement.

Start (Request 2): This command is used to ask a node to start the handshaking procedure with a node (Section 3.13). The payload contains the channel, the generalized address of the new peer, and

a list (eventually empty) of attributes (see Section 3.11) such the PUNCTURING and the ROUTING_PROBABILITY.

Stop (Request 3): Is used to ask a peer to stop sending data toward a lower-peer. The payload contains the channel and the Peer_ID of the lower-peer.

Redirect (Request 4): It works like a stop followed by a start command, but it works as an atomic command (this grant resources availability). The payload contains the Peer_ID and the channel of the old peer, the generalized address of the new peer, and the new channel. There is also a list of attributes such the PEER_CREDENTIAL, the PUNCTURING and the ROUTING_PROBABILITY.

Closing (Request 5): This request is used to communicate to a lower peer the intention to stop the transmission of one or more channels. The payload is the concatenation of the channel numbers that will be closed, therefore if there is no payload, all the channels will be closed. Additionally, if the node does not receive the acknowledgement of this packet, it may close the transmission anyway.

Hello (Request 6): This packet is used in the handshaking phase to transport a cryptographic certificate with the information needed for creating and/or verifying the sender signature. This packet does not require the senders signature field (because it cannot by verified again), and it is considered valid if the payload contains a valid certificate.

Open_Connection (Request 7): This request requires the node to start a CEP toward the peer. The payload is the concatenation of the Peer_ID of the new peer and its generalized address.

Feedback (Request 8): This packet is sent by the lower-peer to give the upper-peer a feedback about its reception statistics for the congestion control described in Section 3.9. The payload, shown in Fig. 3.8 contains the 32 bit timestamp of the last received data packet, the processing delay, the reception rate in packets/RTT, and the estimated loss event rate. This request does not require acknowledgement.

Table 3.1: Values for the Result field of the Acknowledge packet.

Name	Value	Explanation
OK	0	The request was processed successfully
No Resource	1	It was not possible to satisfy the request for lack of resources (e.g., upload bandwidth)
No Reply	2	An handshaking procedure did not complete because no Acknowledge was received to a Set_Parameters request
Bad Target	3	It was requested to stop the data streaming to a node that is not a lower peer

3.8.2.2 TLV format

The TLV format is an easy way to create a list of different kind of data. The format, depicted in Fig. 3.9, is composed by three fields: the first indicates the information *type*, the second is the *length* of the third field which contains the information or *value*.

This is useful because in a list of TLV elements it is possible to easily read each element and also skip some of them by simply reading the type and the length of each element.

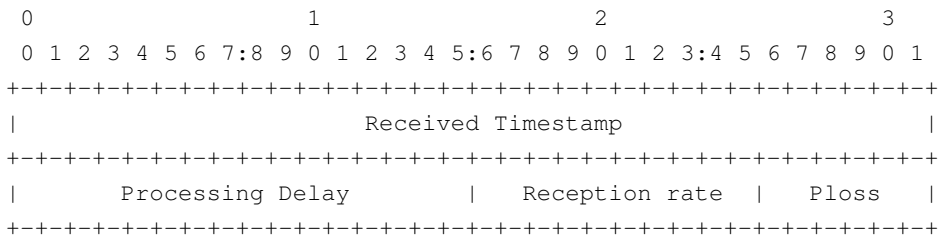


Figure 3.8: Payload of the feedback request.

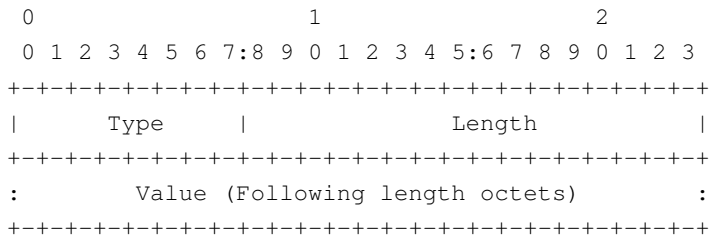


Figure 3.9: TLV format.

3.8.2.3 Control packets retransmission

With data packets, the peer that has sent them does not care if the packets have been received by the lower-peers. With control packets, instead, the peer must be aware of the reception and execution of the command. For this, a command packet must be acknowledged after its execution. Normally, when a peer sends a control packet, it waits for an ACK packet. The ACK packet is sent by the peer that had received the control packet after its execution. Let us call A the peer that had sent the packet, and B the peer that should execute it. After A has sent the packet, it waits for a determinate amount of time for the ACK by B; the control packet has a certain sequence number and the SSN field has value of zero. If A does not receive the ACK before the scheduled timeout, it re-sends the same packet, but with the SSN incremented by one. These operations are repeated a predetermined number of times, after which A considers B not reachable. On the other side, when B receives the packet, executes it, than sends the ACK to A and saves its sequence number and SSN (to avoid to execute twice the same command, and to not acknowledge it too many time). It is possible, due to packets duplications, or a loss of the previous ACK packet, that a command arrives more than once, in this case the peer does not execute it but instead re-sends the ACK packet. If the packet is acknowledged too many times or it is too old, with respect the actual SEQUENCE NUMBER, it is ignored.

3.8.2.4 Control packets elaboration

The first thing that a peer controls when it receives a control packet (and also for the data packets) is the SENDER SIGNATURE[10]. If the signature is not valid, the peer discards the packet. Otherwise, it controls the R flag, and if this flag is set it means that this packet is routed, and the following elaborations are explained in Section 3.8.3.1. If the flag R is not set, the peer checks the SEQUENCE NUMBER and the SSN that must be unique. If these fields are positively checked, it elaborates the content of the payload, and at the end sends an acknowledgement packet to the command source.

[10]If it is required by the security rules in the configuration.

3.8.3 Routed control packets

In some circumstances, either the network manager or a central server must send control packets to a peer. If the peer is behind a NAT, this operation can be not easy. A trivial solution is that the peer continually sends a SET_PARAMETER packet to the server for a predetermined amount of time to keep the NAT hole open, but this solution could pose scalability problems. The solution proposed by PPETP is to use the overlay to propagate the command to the desired peer, as visible in Fig. 3.10. In the upper side of the picture, a server that tries to send a control packet to a client is shown, but this action is blocked by the NAT/firewall of the client. For this reason, as shown in the lower part, it sends the control packet as a routed packet, together to the stream. In this way, are the peers of the network that propagate the command through the NATto the client, that will send an ACK message to the server.

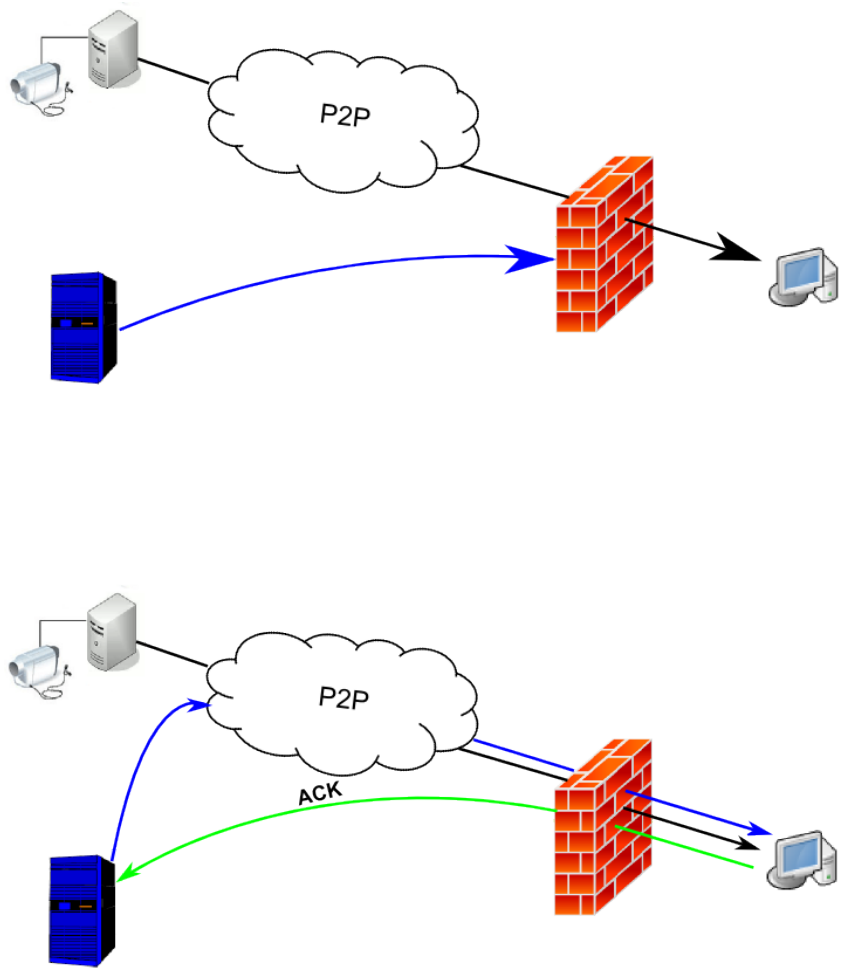


Figure 3.10: Use of routed packet.

To carry out this operation, the R flag of the control packet (see Section 3.8.2) is set and its payload has the structure reported in Fig. 3.11.

In this structure are present the Peer_ID of the peer recipient of the packet, the generalized address of the packet's source where the acknowledgement must be sent, and the Peer_ID. There can be present also the signature (not mandatory) of the packet and the payload of the command.

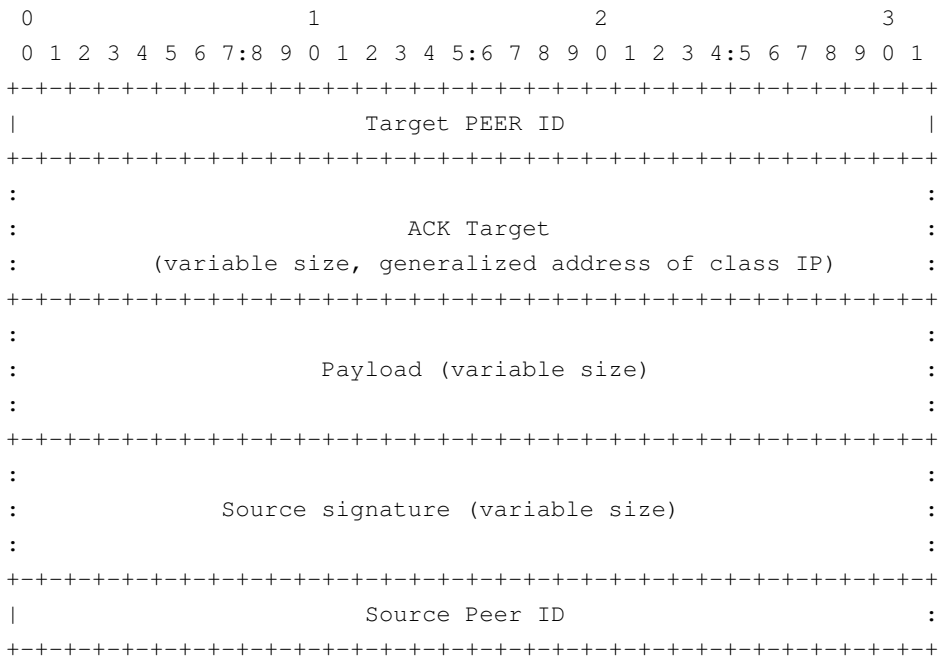


Figure 3.11: Payload of a routed packet.

3.8.3.1 Routed packet elaboration

If the flag R of a control packet is set the peer must execute a more complex control of the packet payload, because it contains other information that must be checked before the execution of the command. The operation successive at the control of the R flag is the control of the fields of the payload. The first thing to check is the SOURCE PEER_ID and the SOURCE SIGNATURE to be sure the packet was generated by a valid source. If not, the packet is discarded. If it is a valid packet the next fields are the number and the SSN, to verify that it is not a duplicated packet, and the last thing is the TARGET PEER_ID, that indicates which peer must execute the command. If this field contains a Peer_ID different from that of the peer, the packet should be propagated to the lower-peers. Hovever, if the packet is directed to this peer, the command contained in the payload must be executed and the acknowledgement must be sent to the address indicated in the field ACK TARGET.

As said, the packets that are not directed to this peer are sent to the lower-peers, and at a first glance this flooding of packets can seem to be a huge waste of bandwidth. In the reality, it is not for the following reason:

- It is expected that the rate of the routed packets, is much smaller than the rate of the data packets hence the increase of the total load is expected to be minimal.
- If a peer receives the same packet twice (checked by the SEQUENCE NUMBER and SSN), it does not propagate it again.
- The first checked field is the SENDER SIGNATURE, therefore only valid sources can create packets that are propagated in the network. It is expected that valid sources do not sends routed packet if they are not needed.
- It is possible control the propagation of the routed packets setting for each peer in the configuration phase, a certain ROUTING_PROBABILITY to each lower-peer[11]. In this way it is possible to create

[11] By default is setted to 1.

a “routing network” that must be a connected to a sub-graph of the actual PPETP network, to ensure that every peer is reachable.

3.8.3.2 Use of reflectors

In the explanation of the routed packets, it was considered that the server that sent the routed packets has a public IP address, but this is not always the case. For some particular configurations it is possible that some users should also be able to send routed packets, even if they are behind a NAT. In this case, the peer is able to send routed packets, but not for receiving the ACK from the destination peer, as depicted in the upper side of the Fig. 3.12. This is because the NAT outgoing command is sent toward a peer of the network different from that sending the ACK, hence this scheme cannot work. To make this

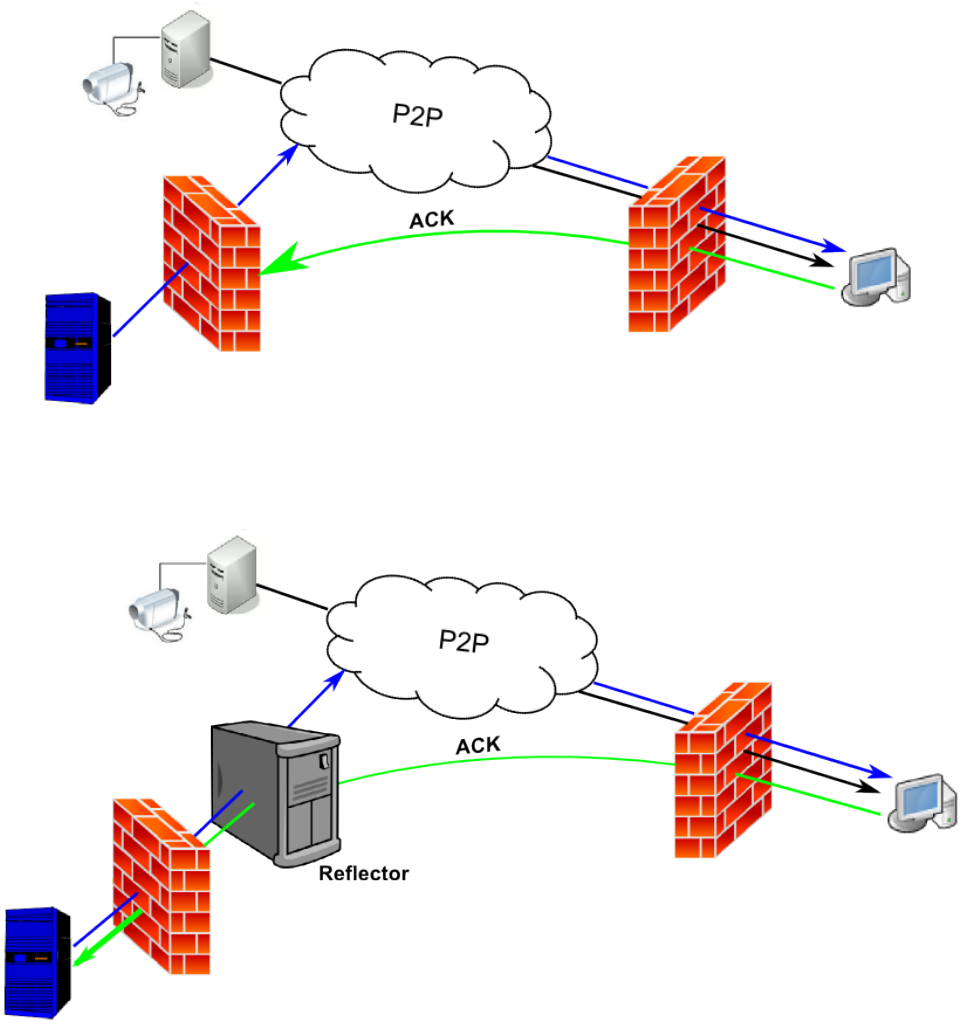


Figure 3.12: Use of reflectors.

operation possible, a solution is the use of a server with a public IP that is used to “reflect” the routed packets and the ACK, shown on the lower side of the Fig. 3.12. The peer that needs to send the routed packet sends the packet to the reflector (in this way it open a hole in its NAT toward the reflector server). The reflector than changes the field ACK TARGET of the routed packet (see Fig. 3.10), putting inside its address, sends the packet, and then waits for the acknowledgement from the recipient of the packet. When the acknowledgement arrives, it can send it to the original peer, and this is possible because the

hole in the NAT is supposed to be still open.

3.9 Congestion Control

In most of cases, is implemented over UDP, an unreliable and connectionless service without rate control. This means that packets can arrive out of order, duplicated, or do not arrive at all. Moreover, the packets rate depends on the applications and is limited only by the speed of the network connection. In presence of network congestion or packets loss, there is no mechanism that informs the host to slow down the packets rate, and this concur to increase the congestion of the network. For this reason, the IETF in the Request for Comments (RFC) "Unicast UDP Usage Guidelines for Application Designers" [51] promotes the utilization of a rate control mechanism when UDP are heavily used. PPETP implements the TFRC: «*TFRC is a congestion control mechanism for unicast flows operating in a best-effort Internet environment. It is reasonably fair when competing for bandwidth with TCP flows, but has a much lower variation of throughput over time compared with TCP, making it more suitable for applications such as streaming media where a relatively smooth sending rate is of importance*» [52]. To work, the TFRC needs some information to determine the characteristics of the network. This can be extracted by the data packet in the fields TIMESTAMP and SEQUENCE NUMBER and by the control packet FEEDBACK. As it is possible to see in [52], this information allows for execution of the rate control's algorithm.

3.10 Puncturing

The *Punturing* is a technique that allows fine control of the upload bandwidth of the peers. Very often, the only reduction factor given by the reduction profile does not allow to exploiting of the entire upload bandwidth of the user, because its granularity is very broad[12]. When the reduction profile allows for some packets loss, for example when some redundancy of packets is granted, a peer can chose not to send all the packets to its lower-peers. This is known as puncturing, a programmed "loss" of packets, made by upper-peers. Obviously, by not sending all the packets, there is a decrese in the uploading bandwidth with respect to that requested by the reduction profile. In PPETP, there are two typologies of puncturing that can be applied to every peer in every channel and for every priority class:

Probabilistic (mode=0) With probabilistic puncturing, each lower-peer is associated a packet-loss probability. The packets that should be sent to the peer are discarded according to this loss probability.

Deterministic (mode=1) With deterministic puncturing, otherwise, packets are discarded according to their sequence number. Each peer is associated with a set of 8-bit numbers. At configuration time, a set $\{m_1, m_2, \ldots, m_L\}$ and a number M are specified; calling N the sequence number of a packet: if $(N \ mod \ (M+1)) \in \{m_1, m_2, \ldots, m_L\}$ then the packet is sent.

3.11 PPETP Attributes

As introduced in Section 3.8.2, some commands include one or more parameters embedded in the TLV format (see Section 3.8.2.2). These parameters are called *Attributes*. Currently, the following attributes are defined:

[12] Only for very high values of the reduction factor this only parameter can be sufficient for a fine control of the bandwidth, but not ever increasing this parameter is a good choice.

Peer_Credential (Type=0) This attribute is used to transmit the information that the upper peer needs in order to sign the packets for the new lower peer. The value of this attribute is a credential certificate whose format is shown in Fig. 3.13.

Punturing (Type=1) This parameter is used to specify the parameters for the puncturing operations. (Section 3.10). The format of the attribute is reported in Fig. 3.14.

Routing Probability (Type=2) It is used to specify the routing probability, and has the same format as the probabilistic puncturing attribute (Fig. 3.14).

Reduction parameters (Type=3) This is used to specify the reduction parameters of a channel. In the first byte, the channel number is reported, while the rest is an opaque value that must be interpreted by the reduction profile.

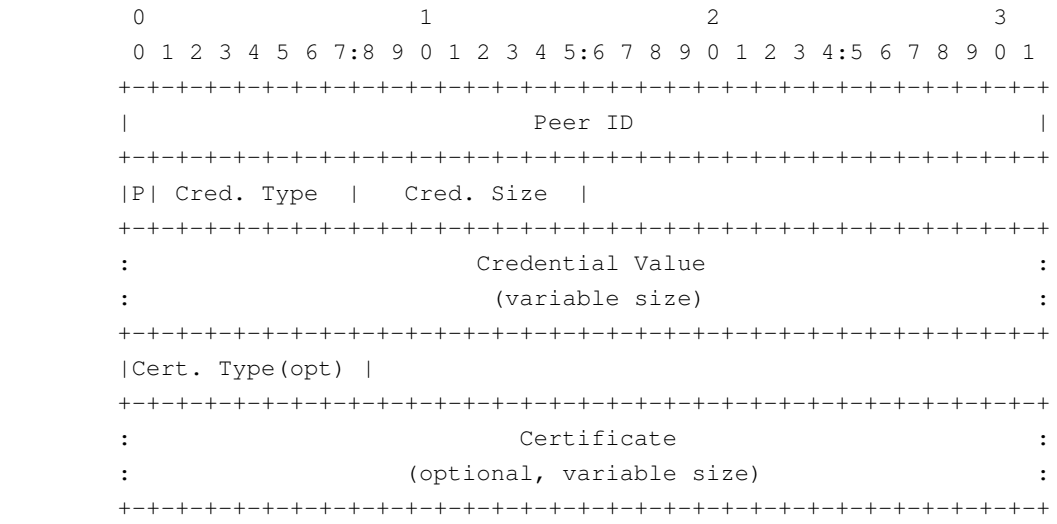
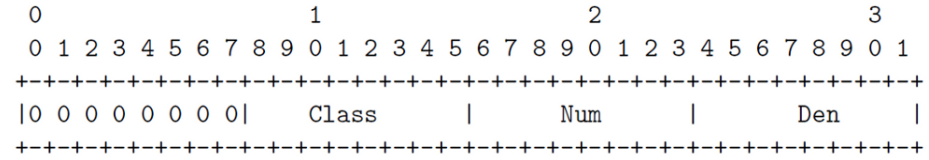
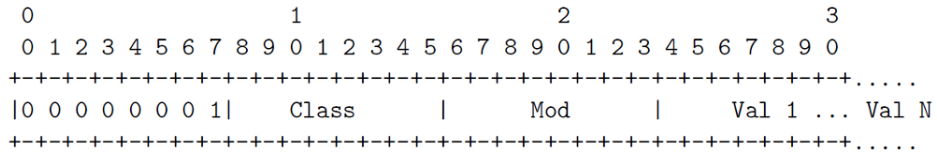


Figure 3.13: Format of credential certificate .



(a) Deterministic



(b) Probabilistic

Figure 3.14: PUNCTURING and ROUTING ATTRIBUTES format.

3.12 Packets processing

3.12.1 Control packets transmission procedure

All of the control packets (with the exception of the ACKNOWLEDGE and FEEDBACK packets) require acknowledgement; if this packet does not arrive, the peer must assume that the command has not been processed. After the transmission of a command, the peer sets a timeout period within which the acknowledgement must be received. Three cases can occur:

1. A **positive** ACK is received before the timeout: the command was successfully executed.
2. A **negative** ACK is received before the timeout: the command was received but the procedure terminates with a failure.
3. **No** ACK is received before the timeout: the peer cannot know if the command or the acknowledgement was lost. In this case, the same command with the same sequence number, the sub-sequence number incremented by 1 is sent again, and a new timeout is set. If the number of retransmission reaches a threshold the procedure terminates with a failure. The transmission timer must be set according to [55].

3.12.2 Control packets acknowledge procedure

On the other side of the transmission, a peer that must acknowledge a received packet must follow the following guidelines:

- The node must send the acknowledgement only after the packet was processed.
- If the received packet has the same sequence number and SSN of an already acknowledgeed packet, it must send another acknowledgement, but it must not process it again.
- Packets whose sequence number is too old respect the most recent sequence number, or those which were acknowledged too many times must be ignored.

3.13 Peer handshaking

In order for two peers to be able to communicate, the following operations, shown also in Fig. 3.15, are necessary:

0 If the generalized addresses of the nodes are not IP addresses, a CEP procedure must be executed in order to obtain an IP address that can be used to communicate with the other peer.

1. The lower-peer sends an HELLO packet with its credentials (if needed) to the upper-peer. This packet is not signed but it is considered valid if the carried certificate is valid.
2. The upper-peer sends the acknowledgement for the HELLO. This packet is signed, but the lower-peer cannot verify it, because it does not know again the sign of the upper-peer, therefore it waits for the upper-peers HELLO before accepting this packet.
3. The upper-peer, sends its HELLO to the lower-peer, and with the reception of this packet the lower-peer can verify the validity of the previous ACK packet.
4. The lower-peer sends the acknowledgement to the received HELLO.

5. When the upper-peer receives the ACK, it sends the packet SET_PARAMETER with the reduction profile parameters.
6. When the ACK for the SET_PARAMETERS is received, the upper-peer can start streaming.

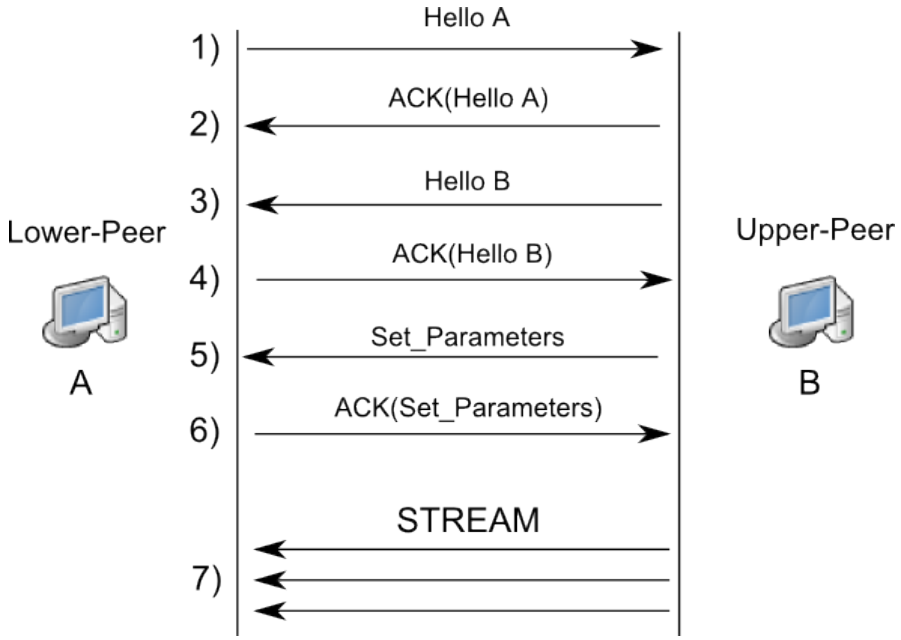


Figure 3.15: Handshake procedure.

Chapter 4

PPETP Details

After the presentation of the main characteristics of PPETP in the previous chapter, here some other features will be shown. Particularly, a protocol for the configuration of PPETP is presented. This is not mandatory for the protocol's operation, but it was designed as a common tool for its configuration, by means of a configuration authority. This protocol can facilitate the diffusion of PPETP because it allows different software[1] to work together using a standard protocol.

4.1 Security considerations

The PPETP project was realized by taking into account some security considerations. Being a distributed protocol, all critical operations should be expressly authorized before being performed. Otherwise, malicious users could compromise the correct functioning of the protocol. To allow an easy deployment of applications based on this protocol, it is possible define the security rules that need to be applied in the configuration procedure. It is possible to develop applications without any security rules, until we arrive at a scenario in which control, data packets, and peers must all be authenticated.

4.1.1 Poisoning attack

In this kind of attack, a peer (or more than one in a coordinated attack) sends to another peer "bogus" data packets that are useless for the reconstruction of the content packet. If not discovered, this method allows a corrupted stream to be propagated by a large set of peers through the P2P mechanism. If the data is authenticated, it is possible to discover this attack after the reconstruction, and chose not to propagate the data again. Using a NC profile, it is possible to do something more, receiving the stream by a number of upper-peers greater than the minimum requested for the reconstruction. In this condition, after an erroneous reconstruction it is possible to try to do it again by changing the used packets. With this technique, it is also possible to discover which peer sent the corrupted packet.

4.1.1.1 Large bandwidth nodes

Considering the case where a node N has a large upload bandwidth with many channels, it is possible that a peer U is completely fed by N. In this case, N can produce an arbitrary stream that will be correctly reconstructed by its lower-peer. The poisoned stream is then propagated to the lower-peer of U, but this, if it is also fed by other upper-peer, will identify the corrupted packets.

[1] E.g., it is possible use a configuration server and a player developed by different teams.

4.1.2 Multiple stream session

A different type of poisoning attack is when a node injects on the session packets belonging to a different stream. In this case, the victim does not recognize the attack, since the packet arrives from a single source only. In order to avoid this attack, it is important to specify the ID of the allowed streams in the security policies.

4.1.3 Defamatory attack

When a peer produces malicious packets and it is discovered, in some contexts it could be notified to a system authority, that can ban the peer from the network. In this situation, a defamatory attack is possible, that is the poisoning of the stream while pretending to be another user. To prevent this attack with PPETP it is possible to request that an upper-peer sign every data packet using a secret key, shared in the configuration phase.

4.1.4 Security Model

As seen in Section 3.8.2, there are some particular kinds of commands that are used to control some important characteristics of the network. These sensitive actions are:

- Sending data-flow control commands (Start/Stop/Redirect).
- Send third party data-flow control commands. A third party control packet is a packet sent by a peer that is not the target of the command. For example, in the commands START and STOP it is possible to specify which peer is the target of the command.
- Sending routed packets.

Associated with these capabilities PPETP defines the same capabilities, partitioned in classes:

1. Self data-flow control class:
   - SELF_START.
   - SELF_STOP.
   - SELF_REDIRECT.
2. Third party data-flow control class:
   - 3RD_START.
   - 3RD_STOP.
   - 3RD_REDIRECT.
3. Routing packets class.

In the configuration phase, one peer can be assigned zero, one or more than one of the capabilities above.

4.1.4.1 Node classes

As said, in the configuration phase the capabilities of the privileged peers are specified. To do this without an explicit declaration, a set of the Peer_ID are reserved. The initial segment $1, 2, \ldots, 2^L - 1$, of the Peer_ID space is reserved for privileged peers, while the remaining IDs are for non-privileged peers. By default, $L = 10$, but it can be modified. The first most significant bits denote the class of the peer, while the other bits identify a specific peer in the class. More details are reported in [25].

4.2 PPETP Configuration

In order to join a PPETP session a node needs some informations for the configurations of the parameters of a session, for example information regarding the reduction profile used for the whole session, the channels, and some information regarding security, such as sender and source signatures algorithm and parameters, privileged classes peers and credentials. Moreover, these information, a peer needs to know its upper-peers hence the configuration contains a list of them, or enough informations to find them (e.g., a method to contact a DHT to be queried).

4.3 Bootstrap configuration protocol

4.3.1 Address of a PPETP session

Since a PPETP session is a distributed object, it makes little sense to refer to it with the classic IP address. For compatibility with currently available protocols (e.g., SDP [56]) it could be useful refer to a session with a <pseudo-address,pseudo-port> pair compatible with the common <IP address, port> pair. In many cases the configuration is automatically downloaded from a server by a *configuration protocol*. Therefore, the pseudo-address of a PPETP session is the IP address of the configuration server and the pseudo-port indicates a specific session on that server. The address of a PPETP session is therefore defined as the pair <IP address,session_ID>, where the *session_ID* is a 16 bit unsigned integer.

4.3.2 Design goals

The design of the configuration protocol was thought to satisfy some requirements:

- Must allow user authentication.
- Must be light-weight and suitable to a stateless implementation on server.
- For complex configuration needs, the server should be able to redirect the user to an alternative configuration protocol (that is why it is called “Bootstrap configuration protocol”).

The typical dialogue between a node and the server is expected to be similar to this:

1. The client sends a query to the server with the session number (often the address of the server and the session number is founded in a Session Description Protocol (SDP) file [56]).
2. If the server requires client authentication, it sends a reply with an “UNAUTHORIZED” error code.
3. The client repeats the request, but this time it includes its credentials.
4. The server checks the credentials and, if satisfied, sends back the configuration information. The reply can assume two different forms:
   (a) In the simplest cases the configuration data can be included in the payload of the reply.
   (b) In more complex cases (for example, if some negotiation is required) the reply will redirect the client to use a different server and/or a different configuration protocol.

The motivation for the redirection toward a configuration server is to avoid DoS attacks, because a complex protocol requires the allocation of resources to store the status of a transaction. With this double server approach only the authorized clients are redirected to the configuration server.

4.3.3 Protocol structure

The configuration protocol has a query/response paradigm. The node that wants to join a PPETP session sends a query message to the configuration server and the latter responds with a response message. Both query and response packets are composed of a 32 bit header and a list of zero or more attributes in the TLV format similar, but slightly different, from that of Section 3.8.2.2. The first octet denotes the type, the length is a 15 bits integer encoded in one or two bytes as described in Section 4.3.6.2, and the successive LENGTH bytes is the attribute.

4.3.4 Query packet

The first type of configuration packets are the *Query Packets*. They are composed of a header followed by a list, possibly empty, of attributes. The header of the query packet is depicted in Fig. 4.1. It is composed of:

Session_ID Contains the Session_ID (see Section 4.3.1) of the desired session.

Query_number Is a sequence number that uniquely identify a query. The same number is inserted in the response packet.

V It is the minimum version of the protocol understood by the client and the server. If the server version is unknown (because is the first request that the client does), it is the version of the client protocol.

Magic It is used to distinguish the configuration protocol packets from the other packets.

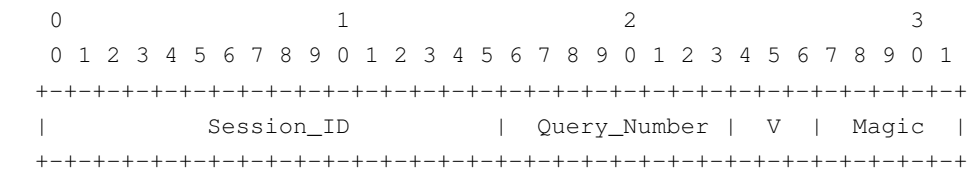


Figure 4.1: Header of a query packet

Query packets are sent using the same port used by PPETP, in this way the remote server can learn the socket address of the PPETP session. If the node is behind a NAT, it inserts the SOCK_ADDR attribute (see Tab. 4.2) into the request. Normally the configuration protocols packets are sent to the TBD port of the server, but this can be changed for example using the PPETP-CONFIG-PORT attribute in the SDP file.

4.3.5 Response packet

The latter type of configuration protocol packets are the *Response Packets* that are used to reply to a query packet. The structure of the header, as shown in Fig. 4.2 is very similar to the structure of the query packets Fig. 4.1. The main difference is the substitution of the SESSION_ID field with

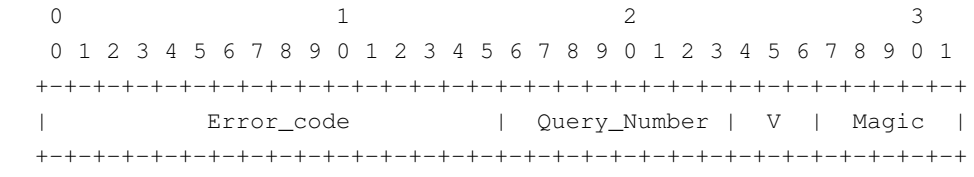


Figure 4.2: Header of a response packet.

the ERROR_CODE field. This field contains a number that indicates the correct or incorrect execution of the query that QUERY_NUMBER is reported in the response packet. The possible values of the ERROR_CODE field are reported in Tab. 4.1 and the complete description is reported in [25].

Table 4.1: Values for the Result field of the Acknowledge packet.

VALUE	MEANING
200	OK
300	Try alternate
400	Bad request
401	Unauthorized
406	Non acceptable
420	Unknown attribute
438	Stale nonce
500	Internal server error

4.3.6 Attributes

The query and the response packet can both carry attributes, and there are many attributes possible, so for the sake of semplicty they are reported in the single Tab. 4.1.

Table 4.2: Attributes of the bootstrap configuration protocol.

Value	Name	Meaning
0	ACCEPTED-PROTOCOLS	A list of integers identifying the version of the configuration protocol implemented by the client
1	PROTOCOL	The protocol that the client must use to get the configuration data
2	PARAMETER	It is a parameter whose meaning depends on the particular protocol specified by PROTOCOL
3	ACCEPTED-CONTENTS	A list of integers identifying a configuration description format understood by the client
4	CONTENT-TYPE	Is an integer that is used when the configuration data is included in the reply
5	CONTENT	The configuration description. The format of this attribute depends on the value of CONTENT-TYPE
6	USERNAME	This field identifies the username and password combination used to generate the signature
7	REALM	It is an unquoted realm-value that matches the grammar "realm-value" as described in [57] but without the double quotes and surrounding white-space

Continued on next page

Table 4.2 – continued from previous page

Value	Name	Meaning
8	USE-NONCE	This field is present when one part requires the other to authenticate itself
9	LOCAL-NONCE	This field is filled with a verbatim copy of the attribute USE-NONCE
10	REMOTE-NONCE	When one of the parts wants to authenticate itself, it MAY add this attribute whose meaning and objective is similar to the "cnonce" field in [58]
11	ACCEPTED-ALGORITHMS	A list of integers identifying a signature computing algorithm that the node (client or server) can use
12	ALGORITHM	Is an integer that specifies the algorithm used to compute the value in the field SIGNATURE
13	USE-ALGORITHM	Is an integer that specifies the algorithm to use in the computation of the value in the field SIGNATURE. If this field is missing, algorithm MAC described in Section 4.3.6.1 is used
14	ACCEPTED-HASHES	Many authentication algorithms make use of hash functions. This attribute is a integer list where each number identifies a hash function that the node (client or server) can use
15	HASH	Is an integer that specifies the hash function used
16	USE-HASH	Is an integer that specifies the hash function to be used in the computation of the value in the field SIGNATURE
17	SIGNATURE	This attribute, if present, MUST be the last one. A packet taht has this field in a different position MUST be discarded and if the packet is a query packet the server must reply with the error code 400. This field is computed by using the algorithm specified in the attribute ALGORITHM
18	REASON	The reason phrase is meant for user consumption, and can be anything appropriate for the error code
19	UNKNOWN-ATTRIBUTES	This attribute is present only in an error response when the response code in the ERROR-CODE attribute is 420

Continued on next page

Table 4.2 – continued from previous page

Value	Name	Meaning
20	SOCK-ADDR	Is a generalized address of class IP and it is used by the client to send the (address, port) pair used to receive PPETP data. By comparing the address found in SOCK-ADDR with the address found in the IP packet, the server can deduce whether the node is behind a NAT or not

4.3.6.1 Packet Signing

The configuration protocol allows both client and server to request the authentication of each other. There are two reasons because a client decides to send signed query:

1. A reply packet with the attribute USE-NONCE was received.
2. Spontaneously. For example if the client receives the nonce in an SDP file.

while a server signs a response if

1. The request packet includes a USE-NONCE attribute and
2. The request packet includes a valid user signature.

To not discourage DoS attacks, the server should not reply with a signed response to a non-signed query.
The procedure for creating a signed packet is the following:

1. A packet signed by the client must contain at least the attribute USERNAME.
2. The value of USE-NONCE (if present) is copied in the attribute NONCE. The value of attribute REALM (if present) is copied in the packet.
3. The Attribute LOCAL-NONCE is added.
4. If necessary, attributes ALGORITHM and HASH are set.
5. The packet, completed with any other attribute related with the query, is processed together the value of USERNAME and REALM to obtain a string of bits. The resulting string of bits is used as value of the attribute SIGNATURE.

HMAC Signature

The specification of the PPETP protocol allows future definition for signature algorithm. In order to grant at least one algorithm, it is specified that this signature algorithm must be implemented in every client and server. The Keyed-Hashing for Message Authentication (HMAC) signature algorithm [59] is based on the knowledge of a secret shared between the client and the server. This secret can be a long-term password or a temporary secret communicated in a secure channel. It is supposed that the shared secret can be found from the knowledge of USERNAME and REALM. Using the notation of [59] "TEXT" is the whole packet to be signed, while the key K is obtained as:

$$K = H(S|NONCE)$$

where S is the shared secret, $NONCE$ is the value of the attribute USE-NONCE, "|" is the concatenation operation and H is the selected hash function.

Algorithm 4.1: Pseudo-code of 15-bit integer encoding.

```
b1: Byte;
b2: Byte;
N: Integer;       # 0 <= N < 32768

Begin
  if N < 128
    b1 := N;      # b2 unused
  else
    b1 := 128 + (N mod 128);
    b2 := int(N/128);
  end
End
```

Algorithm 4.2: Pseudo-code of 15-bit integer decoding.

```
...
Begin
  if (b1 and 128) == 0
    N := b1;
  else
    N = (b1 mod 128) + 128 * b2;
  end
End
```

4.3.6.2 15-bit integers encoding

In many cases, it is necessary to store an integer number into bytes. Very often, these integer numbers have a value smaller than 100, but in some cases they can be larger. The algorithm presented here is an efficient method of storing numbers up to 127 (using only one byte), but allowing values up to 32767 (using two bytes). The algorithm in pseudo-code is reported in the Algorithm 4.1.

In other words, the most significant bit of the first byte indicates whether it is one or two bytes used to store the value. If this bit is 0, only one byteis used, otherwise two byte are used. To retrieve the value, pseudo-code of the inverse algorithm is reported in the Algorithm 4.2.

4.3.7 Compact Configuration Format

In order to configure a PPETP session, the configuration server must provide the new peer with a lot of information. This information can be passed with the *Compact Configuration Format Protocol* that was deigned for this purpose. This protocol is inspired to SDP, but it is designed to be more compact by eliminating some useless characters used by that protocol. The lines that are used in the configuration are the following, where in the square brackets, the mandatory lines with the character "*" are marked, the lines that can appear more than once are marked with "+" and lines that can be followed by an arbitrary number of attributes are marked with "a".

The complete format of these lines, in the Augmented Backus-Naur Form (ABNF) format [60], are reported in [25], while in this section only their meaning is reported.

s	[*]	Stream line
p	[* a]	Profile line
Y	[ ]	Informations about the node itself ("Y" is for "Yourself")
C	[+a]	Informations about the channels opened by the node
c	[+a]	Connection line(s)
a	[+]	Attribute line(s)
f	[a]	Peer search method ("f" is for "find")
n	[+a]	Peer line(s) ("n" is for "node")
k	[+a]	Security related data (e.g., public keys)
P	[+a]	Security policies
X	[ ]	Data puncturing lines
x	[ ]	Routing puncturing lines

Stream line (“s”) The parameters on this line are the identifiers of the allowed streams. If the peer receives a packet with a Stream_ID that is not in this list it must discard it. If the line is missing or if there are no IDs, any ID is allowed.

Profile line (“p”) The first parameter of this line is the name of the utilized profile while the other parameters, if there are any, are defined by the profile. This line can be followed by any number of attribute lines, whose meanings are defined by the reduction profile.

Self information (“Y”) The first parameter indicate the Peer_ID that the node must use or the ’*’ character, meaning that the node can chose its Peer_ID by itself. The second parameter is the number of channels to open; in this case the symbol ’*’ can also appear, meaning that the number of channels is specified by the number of ’C’ lines that follow. If there are no other parameters, the node cannot produce any stream. Otherwise, the following parameters are the Stream_IDs that the peer can produce. There are some possibles methods to specifies the IDs:

1. A list of integer, where every integer is a Stream_ID that the peer can produce.
2. The line can end with a single asterisk, in which case the peer can produce any number of streams whom ID is selected by the peer.
3. The line can terminate with an asterisk followed by an integer; in this case, the number of streams that the peer can produce is limited by the integer.

Output channels (“C”) This line specifies the reduction parameters to be applied to a channel. Similar to the profile line, the parameter, or the following attribute lines are specified by the profile. If the character “C” is followed by an asterisk and a number, it means that there are two consecutive channels with the same parameters.

Connection (“c”) This line specifies a generalized address. It can appear as part of a peer-block or by itself. In the latter case, the field “label” is mandatory. The first parameter identifies the class of the address, allowing the IP and ICE classes to be identified. If the following parameter is a

label, identified by the '@' as the first character, it means that the address can be referred in the remaining of the description simply by the label. The information necessary for the connection can be given as positional parameters or as attributes.

Peer searching ("f") The list of the upper-peers can be specified in the rest of the configuration or it can be obtained using another search method indicated with this line. The first parameter indicate the method used for searching while the other parameters and/or attributes are specified by the method. For example, a method can be a DHT and the other parameters are used to execute a search on it.

Peer line ("n") This line it is used to describe a peer of the node. The first parameter is a character that indicates the peer-type; it can assume the values '*u*' for upper-peers, '*l*' for lower-peer and '*o*' for the other peers. The last case it is used for those nodes that need to communicate with the peer, but are not a lower or an upper peer (e.g., the bridge in the ICE-based NAT transversal). The '*o*' type does not require the channel parameter, while for the '*u*' this parameter indicate the channels of the upper-peer required, and for the '*l*' peers it indicates the channel used for it. After these information there is the generalized address of the peers, as a *connection-body*, or label, or with the '*c*' lines.

Security policies ("P") This line specifies the actions that a peer can do. The first parameter indicates the "capability" and is followed by the list of the Peer_ID of the peers authorized for that capability. Further than the Peer_ID, it is possible to indicate some keywords used to identify the same categories of peers.

Attribute ("a") This line is used to assign parameters in a nominal way. It is composed by the attribute's name, followed by the symbol '=' and then the attribute's value. The interpretation of the value depends on the attribute.

Security related data ("k") Not again specified.

4.3.8 Configuration defaults

The complete configuration of a PPETP session requires many parameters as seen in the previous section. In order to simplify the configuration step and minimize the amount of required data, this section defines some configuration defaults that can provide a good setup for most of the application contexts.

- Data and control packets signed with sender signature described in Appendix A.2.
- Routed control packets require a source signature. The source signature is done with the RABIN procedure described in Appendix A.3.
- Only authorized nodes can send routed packets.
- Only authorized nodes can sign credential certificates.
- Only authorized nodes can send flow control command.
- The session carries only one stream with Stream_ID equal to 1.

4.4 ICE

To allow PPETP to work with client behind devices like firewall and NAT, this version of the protocol implements ICE [54] as its default CEP. The information necessary to execute the ICE's algorithm is usually contained in the generalized address (see Section 3.5.1) or in the configuration file as explained in the Section 4.3.7. The generalized address contains, among other things, the address of a "bridge" node used by the peers to exchange the lists of the candidates. The lists, collected by both of the peers are sent to the bridge by an HTTP/HTTPS request. The protocol implements the JavaScript Object Notation (JSON) [61] format to encode the candidates list, but other formats can be used in future. The complete JSON schema for the candidates list is reported in [25]

The bridge, after receiving both the lists, exchanges the lists amongst the two peers, after that, the check for connectivity can begin. So that this may happen, the bridge needs some other information. This information is the Peer_ID of the requiring and requested peer and the session identifier. This information allows to create a unique association between the peers and the peers lists. Note that a session identifier is also necessary because the same peers can be connected to each other in more than one session. These sessions can also share the same address but not also the same port, for this a new CEP is required.

Conclusions

The study on the delay jitter clearly shows that a P2P network, using NC procedures such as coding technique, allows for a reduction of the jitter experienced by a node. This improvement can significantly increase the Quality of Experience (QoE) of the users applications, because of the possibility of reducing the player's buffer size. This results in a time decrease when the user turns on the software or, in a multi-channel system, when it changes channel. The drawback of the proposed technique is an increase in the overall bandwidth size due to the redundancy introduced by this scheme.

The second topic addressed in the thesis was a study of the asymptotic packets loss probability. It was shown that the equivalent loss probability P_{eq}, experienced by a node of a P2P network employing NC, converges to 1 at the increase of of the network size. However, it is possible to control the convergence velocity, meaning that is possible make it "very slow", while operating on the network parameters. In this way it is possible to make the P_{eq} negligible in networks of typical size without increasing the redundancy.

The main contribution of this thesis was the presentation of PPETP. This protocol allows for the distribution of contents through the Internet in a multicast fashion, exploiting an overlay build on a P2P network. It includes many features such as the possibility to distribute many streams together, that can be produced by a single or multiple sources, different levels of security management, the availability of NAT traversing procedures, and a versatile and extendible information coding technique. The design of the protocol takes into account the possibility of configuring and expanding most of the functionality of PPETP, making it a very versatile protocol adaptable to many applications. The best thing about PPETP is the avoidance for the applications that use it, of most of the aspects of network management and data flow. In this way, it appears similar to a transport protocol, making the development of new applications and the adaptation of already existing ones easier, while giving them the possibility to exploit a P2P architecture.

Appendix A

PPETP builtin profiles

PPETP demands some duties for several "plugins" (e.g., reduction and signature profiles, NAT traversal procedures) whose definition is not part of the PPETP "core". In order to make PPETP usable without waiting for the definition of all the necessary plugins, this section defines few basic reduction and signature plugins.

A.1 Reduction profiles

A.1.1 How to define a reduction profile

To specify a reduction profile, it is necessary to specify at least:

- The profile name and the name and type of every profile parameter.
- Which reduction parameters are "global" to the whole PPETP session and which are "local" to each peer.
- The algorithm to map a content packet into the data packet payload.
- The format used to store the parameters in the payload of the SET_PARAMETER request and in the payload of a data packet if the flag INLINE is setted.
- The meaning of the FLAGS field in the data packet.
- Any reduction-profile specific request.

A.1.2 Basic reduction profile

This is a very simple profile that just copies the content packet in the payload. It can be used to distribute streams with a low bitrate (e.g., RTP Control Protocol (RTCP) [62] streams).

A.1.2.1 Profile name and parameters

The name of the profile is BASIC and there are no other parameter to specify.

A.1.2.2 Payload construction

The payload is a verbatim copy of the content packet.

A.1.2.3 Profile-related definitions

- Data packet flags: Flags F, G, H and I are unused.
- SET_PARAMETERS payload: SET_PARAMETERS carries no payload.
- Profile-specific request: This profile defines no profile-specific request.

A.1.3 Vandermonde reduction profile

A.1.3.1 Profile name and parameters

The profile name is VANDERMONDE. This profile defines the following parameters:

gf_size This parameter can assume the values 1, 2 and 4 and determines the size of the Galois field used. More precisely, GF_SIZE is the size in octets of an element of the Galois field, therefore the Galois field relative to GF_SIZE is $GF(2^{8 \cdot gf\_size})$.

reduction-factor This is (approximately) the ratio between the size of a content packet and its reduced version. This value was called R in Section 3.6.1.

reduction-base This is the element of $GF(2^{8 \cdot gf\_size})$ used to construct the reduction vector. This value was called b in Section 3.6.1.

The parameters GF_SIZE and REDUCTION-FACTOR are global for the whole PPETP session, while the value REDUCTION-BASE is local to each node. Depending on the configuration, the REDUCTION-BASE can be chosen autonomously by the peer or it can be imposed by some external entity.

A.1.3.2 Payload construction

The payload construction is based on the ideas of [40]. The payload is constructed as follows

1. Define, for the sake of compactness, $d = 8 * gf\_size$, $B = reductionbase$ and $R = reduction\text{-}factor$.
2. Let the elements of $GF(2^d)$ be represented as described in Appendix A.1.3.2.1.
3. At startup the node constructs the row vector $r = [1, b, b^2, \cdots, b^{R-1}]$.
4. The packet to be reduced is mapped in a matrix G with R rows and $L/(gf\_size * R)$ columns with entries in $GF(2^d)$ as follows:
   (a) The packet is padded, as described in Appendix A.1.3.2.2, to a length multiple of $gf\_size * R$ octets. Let L be the length, in octets, of the padded packet.
   (b) Let $b[n]$ be the n-th octet of the padded packet, with $n = 0$ denoting the first octet. For every $m = 0, 1, \ldots, L/gf\_size$, interpret the sequence of GF_SIZE octets: $b[gf\_size * m], b[gf\_size * m + 1], \ldots, b[gf\_size * (m + 1) - 1]$ as an element of $GF(2^d)$ as described in Appendix A.1.3.2.1. Let $g[m]$ be the element of $GF(2^d)$ associated to $b[gf\_size * m], b[gf\_size * m + 1], \ldots, b[gf\_size * (m + 1) - 1]$.
   (c) Define G as the matrix whose element in row r and column c is $g[r + R * c]$, where $r = 0, 1, \ldots, R - 1$ and $c = 0, 1, \ldots, L/(R * gf\_size) - 1$.
5. Matrix G is left-multiplied by vector r to obtain row vector $U = r * G$.

6. Every element of U is mapped to gf_size octets (still according to the representation described in Appendix A.1.3.2.2) to obtain a string of L/R octets that represents the payload of the data packet.

A.1.3.2.1 Galois field implementation If $d = 8, 16$ or 32, let $GF(2^d)$ be the field of polynomials with coefficients in $GF(2)$ (i.e., the integers modulo 2) modulo the polynomials shown in Tab. A.1

Table A.1: Polynomials used to define $GF(2^d)$.

d	Polynomial defining $GF(2^d)$
8	$x^8 + x^4 + x^3 + x^2 + 1$
16	$x^{16} + x^5 + x^3 + x^2 + 1$
32	$x^{32} + x^{15} + x^9 + x^7 + x^4 + x^3 + 1$

The element of $GF(2^d)$ associated with

$$c_{d-1}x^{d-1} + c_{d-2}x^{d-2} + ...c_1x + c_0$$

(where each $c_n = 0, 1$) is represented by the d-bit unsigned integer

$$C = 2^{d-1}c_{d-1} + 2^{d-2}c_{d-2} + ...2c_1 + c_0$$

This integer can be represented as a sequence of octets $b_0, b_1, b_{d/8-1}$ in little endian order, that is

$$C = b_0 + 256b_1 + 256^2b_2 + \ldots$$

A.1.3.2.2 Packet padding

1. Let LENGTH(P) be the size in octets of the content packet P to be padded and let the padding length L be
$$L = (gf\_size * R) - (\mathsf{length}(P) \mod (gf\_size * R))$$

2. Note that $L + \mathsf{length(P)}$ is always a multiple of $R * gf\_size$. Note also that if LENGTH(P) is already a multiple of $R * gf\_size$, the packet will be padded with $L = R * gf\_size$ bytes, although no padding would be necessary. It was chosen to add the padding also when LENGTH(P) is already a multiple of $R * gf\_size$ for the sake of simplicity, in order to not handle special cases. The overhead in bandwidth is expected to be negligible (an average of GF_SIZE bytes every $R * gf\_size$ packets, that is, $1/R$ byte per packet).

3. (a) Append L zeros to the packet.
   (b) Decompose L as
   $$L = A * 128 + B$$
   where $0 \leq B < 128$.
   (c) If $A = 0$ (that is, the padding length is less than 128), write B in the last octet of the padded packet.
   (d) If $A > 0$, write $B + 128$ in the last octet of the padded packet and write A in the penultimate octet.

The algorithm above can be summarized by saying that the most significant bit of the last octet of the padding acts as a flag: if it is zero, we know that the padding length was less than 128 and that its value is in the last octet; if it is one, we know that the padding length was greater or equal than 128 and that its value is stored in the last two octets. Note that using only one octet would limit the padding size to 255 and that we cannot always use two octets because the padding size could be 1.

A.1.3.3 Profile-related definitions

- Data packet flags: Flags F, G and H are unused; flag I has its default meaning of "Inline".
- SET_PARAMETERS payload: The payload of the SET_PARAMETERS command is used to transfer the value chosen for reduction-base. Such a value is represented as a sequence of GF_SIZE octet used as the payload of SET_PARAMETERS.
- Payload with the INLINE bit set: If the INLINE bit is set, the value of reduction-base, encoded as explained above, is prepended to sequence of octets resulting from the reduction procedure. The result is the payload of the data packet.
- Profile-specific request: This profile defines no profile-specific request.

A.2 Sender signature profiles

A.2.1 How to define a sender signature profile

A sender signature profile document must specify at least

- The profile name and name and type of any required parameter.
- Which parameters are "global" to the whole PPETP session and which are "local" to each peer.
- The algorithm to obtain the source signature field from the packet.
- Any profile-specific request.

A.2.2 Shared key signature profile

A.2.2.1 Profile name and parameters

The name of this profile is SHARED-KEY. This profile requires the following parameters:

- An h-bit hash function H, at least SHA-256 must be supported. The name of this parameter is HASH. The only value currently accepted for hash is SHA-256, but other values can be added in future.
- A symmetric encryption algorithm C, at least AES-256 MUST be supported. The name of this parameter is ENCRYPTION. The only value currently accepted for encryption is AES-256, but other values can be added in the future.
- Two positive integers ID-SIZE and MAC-SIZE, both multiple of 8 and with MAT-SIZE $\leq h$.

The nodes agree on these parameters via extra-PPETP means. A summary of the parameters together with the accepted values is given in Tab. A.2.

Table A.2: Configuration parameters for the shared key signature profile.

Parameter	Attribute name	Accepted values
Hash function	HASH	SHA-1
Encryption function	ENCRYPTION	AES-256
ID size in octets	ID-SIZE	non-negative integer ≤ 8
MAC size in octets	MAC-SIZE	non-negative integer ≤ 16

A.2.2.2 Payload construction

This signature profile supposes that the sender and the receiver share a common secret K. The sender is identified by an ID represented by a ID-SIZE-bit number. The signature of a packet is the pair (ID, MAC), where MAC is calculated as follows

1. The whole packed is processed with hash function H.
2. The result of the hash is encrypted with C using K as key.
3. The first MAC-SIZE bits of the encrypted hash are the MAC.

The signature payload is obtained by linking the ID-SIZE-bit number representing the ID and the MAC-SIZE-bit number representing the MAC. Since both ID-SIZE and MAC-SIZE are a multiple of 8, the signature will always take an integer number of octets.

A.2.2.3 Remarks

- This profile does not say how the two nodes agree on the common secret K, this is supposed to be done via extra-PPETP means. For example, if the two nodes are a server and a user, K could be a long-term password, whereas if the two nodes are two peers K could be the result of a Diffie-Hellman key agreement procedure.
- In order to ensure that the packet was signed by a node A, it is necessary to be sure that both ID and K refer to A. This will typically require some form of authentication that must be done via extra-PPETP means.
- In order to allow for an autonomous Diffie-Hellman key exchange between the nodes without involving a central server, a node can communicate its ID and its public Diffie-Hellman key in the PEER_CREDENTIAL attribute of the DATA_CONTROL/START packet.
- The possibility of having the MAC shorter than the hash allows for a reduction of the bandwidth required by the signature in those applications that do not need the strength of the full MAC.

A.2.3 Void signature profile

This profile does not add any signature to the packet. It is defined for those cases where signatures would be redundant.

A.2.3.1 Profile name and parameters

The name of this profile is VOID. This profile defines no parameters.

A.2.3.2 Creating the signature

This profile does not create any signature. The payload is empty.

A.2.3.3 Verify the signature

The signature check is always positive.

A.3 Source signature profiles

A.3.1 How to define a source signature profile

A source signature profile document must specify at least

- The profile name and name and type of any required parameter.
- Which parameters are "global" to the whole PPETP session and which are "local" to each peer.
- The algorithm to obtain the source signature field from the packet.
- Any profile-specific request.

A.3.2 Rabin signature profile

This profile is based on the Rabin signature algorithm [63].

A.3.2.1 Profile name and parameters

The name of this profile is RABIN. This profile defines the following parameters:

- A parameter SIGN-SIZE assuming positive values less than or equal to 16.
- A parameter TAIL-SIZE assuming positive values less than or equal to 8.

A.3.2.2 Creating the signature

The procedure to compute the source signature is the following:

1. The procedure is parametrized by two positive integer values: $s \leq 16$ and $u \leq 8$.
2. At the beginning the node generates two $4*$SIGN-SIZE-bit prime numbers p and q (the node private key) and calculates the sign-size-octets value $n = p * q$ (the public key).
3. To sign a packet, the node concatenates the whole routed packet (including the routing data block, but not the signature) with a tail-size-octets random value U and process the result with SHA-256. Let Y be the final value.
4. The node finds x such that $Y = x^2 \mod n$. If such an x does not exist, the node draws a new U, goes back to the previous step and tries again. The expected number of trials is four. Note that the node can find x efficiently because it knows p and q.
5. The signature is given by the (sign-size+tail-size)-octets value $2^{8*tail-size} * x + U$. Such a value is stored in the Source Signature field with any unused most significant bits set to zero.

A.3.2.3 Verifying the signature

The procedure to verify the signature is the following:

1. From the knowledge of the source ID, determine the source public key N. If no public key is associated with the source ID, the verification fails.
2. Extract values x and U from the Originator Signature field.
3. Concatenate U with the packet and process the result with SHA-256 to obtain T.
4. Verify that $T = x^2 \mod n$.

The association of the public keys with the corresponding peer ID is supposed to be done by extra-PPETP means.

Bibliography

[1] C. System, "Cisco visual networking index: Usage study." http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/Cisco_VNI_Usage_WP.html, October 2010.

[2] A. Vakali and G. Pallis, "Content delivery networks: status and trends," *Internet Computing, IEEE*, vol. 7, pp. 68 – 74, nov.-dec. 2003.

[3] R. Buyya, M. Pathan, and A. Vakali, eds., *Content Delivery Networks (Lecture Notes Electrical Engineering)*. Springer-Verlag Gmbh, 1 ed., 2008.

[4] N. Bartolini, E. Casalicchio, and S. Tucci, "A walk through content delivery networks," in *Performance Tools and Applications to Networked Systems* (M. Calzarossa and E. Gelenbe, eds.), vol. 2965 of *Lecture Notes in Computer Science*, pp. 1–25, Springer Berlin Heidelberg, 2004.

[5] ""akamai", http://www.akamai.com."

[6] ""mirror image", http://www.mirror-image.com."

[7] ""webvisions", http://www.webvisions.com."

[8] S. Deering, "Host extensions for IP multicasting." RFC 1112 (Standard), Aug. 1989. Updated by RFC 2236.

[9] B. Williamson, *Developing IP Multicast Networks, Volume I*. Cisco Press, 1999.

[10] F. Pianese, *"A Survey of P2P Data-driven Live Streaming Systems" in Streaming Media Architectures, Techniques and Applications: Recent Advances*, ch. 12, pp. 295–310. Eds. Hershey, NY, USA: Information Science Reference, 2011.

[11] F. Mathieu and D. Perino, *"Epidemic Live Streaming" in Streaming Media Architectures, Techniques and Applications: Recent Advances*, ch. 13, pp. 311–336. Eds. Hershey, NY, USA: Information Science Reference, 2011.

[12] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, and Y. Wang, "LayerP2P: Using layered video chunks in P2P live streaming," *Multimedia, IEEE Transactions on*, vol. 11, pp. 1340–1352, Aug. 2009.

[13] S. Alstrup and T. Rauhe, "Introducing Octoshape – a new technology for large-scale streaming over the Internet," July 2005.

[14] M. Wang and B. Li, "Network coding in live peer-to-peer streaming," *IEEE Transactions on Multimedia*, vol. 9, pp. 1554–1567, Dec. 2007.

[15] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," in *IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems*, pp. 1655–1666, 2007.

[16] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," pp. 298–313, 2003.

[17] X. Zhang, J. Liuy, B. Liz, and P. Yum, "CoolStreaming/DONet: a data-driven overlay network for efficient live media streaming," in *Proceedings of IEEE International Conference on Computer Communications*, IEEE Computer Society, Mar. 2005.

[18] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)." RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[19] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in peer-to-peer live multimedia streaming," in *Proceedings of the 14th annual ACM international conference on Multimedia*, MULTIMEDIA '06, (New York, NY, USA), pp. 539–548, ACM, 2006.

[20] G. Dán, V. Fodor, and I. Chatzidrossos, "On the performance of multiple-tree-based peer-to-peer live streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 2556 –2560, may 2007.

[21] G. Dán, V. Fodor, and G. Karlsson, "On the stability of end-point-based multimedia streaming," in *in Proc. of IFIP Networking*, pp. 678–690, May 2006.

[22] G. Dán, V. Fodor, and I. Chatzidrossos, "Streaming performance in multiple-tree-based overlays," in *in Proc. of IFIP Networking*, pp. 617–627, May 2007.

[23] ""pplive", http://www.pplive.com."

[24] ""ppstream", http://www.ppstream.com."

[25] R. Bernardini, R. Cesco Fabbro, and R. Rinaldo, "Peer-to-peer epi-transport protocol draft-bernardini-ppetp." `http://tools.ietf.org/html/draft-bernardini-ppetp`, July 2011.

[26] I. Chatzidrossos, G. Dán, and V. Fodor, "Delay and playout probability trade-off in mesh-based peer-to-peer streaming with delayed buffer map updates," *Peer-to-Peer Networking and Applications*, 2010.

[27] G. Dán and V. Fodor, "Delay asymptotics and scalability for Peer-to-Peer live streaming," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, pp. 1499–1511, Nov. 2008.

[28] D. Wu, Y. Liu, and K. Ross, "Modeling and analysis of multichannel p2p live video systems," *Networking, IEEE/ACM Transactions on*, vol. 18, pp. 1248 –1260, aug. 2010.

[29] D. Wu, Y. Liu, and K. Ross, "Queuing network models for multi-channel p2p live streaming systems," in *INFOCOM 2009, IEEE*, pp. 73 –81, april 2009.

[30] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 919 –927, may 2007.

[31] G. Dán and V. Fodor, "Stability and performance of overlay multicast systems employing forward error correction," *Perform. Eval.*, vol. 67, pp. 80–101, February 2010.

[32] L. D. Soares, P. Nunes, and F. Pereira, "Efficient network-aware macroblock mode decision for error resilient h.264/avc video coding," SPIE, September 2008.

[33] P. Nunes, L. D. Soares, and F. Pereira, "Automatic and adaptive network-aware macroblock intra refresh for error-resilient h.264/avc video coding," in *Proceedings of the 16th IEEE international conference on Image processing*, ICIP'09, (Piscataway, NJ, USA), pp. 3037–3040, IEEE Press, 2009.

[34] J.-R. Ohm, "Advances in scalable video coding," *Proceedings of The IEEE*, vol. 93, pp. 42–56, 2005.

[35] V. K. Goyal, "Multiple description coding: Compression meets the network," *Signal Processing Magazine, IEEE*, vol. 23, no. 5, pp. 74–94, 2001.

[36] S. Zezza, E. Magli, G. Olmo, and M. Grangetto, "Seacast: a protocol for peer-to-peer video streaming supporting multiple description coding," in *Proceedings of the 2009 IEEE international conference on Multimedia and Expo*, ICME'09, (Piscataway, NJ, USA), pp. 1586–1587, IEEE Press, 2009.

[37] J. R. Taal and R. Lagendijk, "Hybrid temporal-snr multiple description coding for peer-to-peer television," in *Picture Coding Symposium*, 2006.

[38] R. Ahlswede, N. Cai, S.-y. R. Li, and R. Yeung, "Network information flow," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 46, no. 4, pp. 1204–1216, 2000.

[39] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the 41st Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[40] R. Bernardini, R. Rinaldo, and A. Vitali, "A reliable chunkless peer-to-peer architecture for multimedia streaming," in *Proc. Data Compr. Conf.*, (Snowbird, Utah), pp. 242–251, Brandeis University, IEEE Computer Society, Mar. 2008.

[41] Q. Li and D. Mills, "On the long-range dependence of packet round-trip delays ininternet," *ICC 98. Conference Record.1998 IEEE International Conference on Communications, 1998.*, vol. 2, pp. 1185–1191, 1998.

[42] Q. Li and D. Mills, "Investigating the scaling behavior, crossover and anti-persistence of internet packet delay dynamics," in *Proceeding of IEEE GLOBECOM'99, Rio De Janeiro*, pp. 1843–1852, 1999.

[43] Q. Li and D. Mills, "Jitter based delay boundary prediction of wide-area networks," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 578–590, 2001.

[44] Q. Li and D. Mills, "The implication of short-range dependency on delay variation measurement," in *IEEE International Symposium on Network Computing and Applications*, pp. 374–380, 2003.

[45] H. A. David, *Order Statistics 2nd edition*. Wiley-Interscience, 1981.

[46] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*. National Bureau of Standards, 1965.

[47] ""planetlab", http://www.planet-lab.org."

[48] D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis." RFC 1305 (Draft Standard), Mar. 1992. Obsoleted by RFC 5905.

[49] X. Hei, Y. Liu, and K. Ross, "IPTV over P2P streaming networks: the mesh-pull approach," *IEEE Communications Magazine*, vol. 46, pp. 86–92, Feb. 2008.

[50] R. Bernardini, R. Cesco Fabbro, and R. Rinaldo, "Group-based reduction schemes for streaming applications," *ISRN Communications and Networking*, vol. 2011, 2011.

[51] L. Eggert and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers." RFC 5405 (Best Current Practice), Nov. 2008.

[52] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification." RFC 5348 (Proposed Standard), Sept. 2008.

[53] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)." RFC 4340 (Proposed Standard), Mar. 2006. Updated by RFCs 5595, 5596.

[54] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols." RFC 5245 (Proposed Standard), Apr. 2010.

[55] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer." RFC 2988 (Proposed Standard), Nov. 2000. Obsoleted by RFC 6298.

[56] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol." RFC 4566 (Proposed Standard), July 2006.

[57] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol." RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.

[58] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication." RFC 2617 (Draft Standard), June 1999.

[59] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication." RFC 2104 (Informational), Feb. 1997. Updated by RFC 6151.

[60] D. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF." RFC 4234 (Draft Standard), Oct. 2005. Obsoleted by RFC 5234.

[61] E. K. Zyp, "A json media type for describing the structure and meaning of json documents." `http://tools.ietf.org/html/draft-zyp-json-schema-03`, November 2010.

[62] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." RFC 3550 (Standard), July 2003. Updated by RFCs 5506, 5761, 6051, 6222.

[63] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," tech. rep., Cambridge, MA, USA, 1979.

List of Figures

List of Tables

Acknowledgment

First and foremost, I am deeply grateful to my advisor, Prof. Riccardo Bernardini, for his great help, inspiration and the immense knowledge he provided me with these past years. With his guidance I was able to conclude my project successfully, as well as this very important step of my instruction, which was a personal challenge for me as well.

I am equally grateful to my co-adviser, Prof. Roberto Rinaldo, for his continual availability and for the help gave me during these past years as well as for the many opportunities he gave me.

I would also like to show my gratitude to Prof. Fernando Pereira, Prof. Paulo Nunes, and Prof. Luis Ducla Soares for their full support during my time in Lisbon, and for their constructive criticism, which allowed me to expand my research in new and different directions. This is always a precious contribution for people that seek to improve themselves.

I would like to thank all my friends who shared with me the happiest and the saddest moments during these years, while giving me the necessary motivation to continue my efforts. Together with the "old" friends, I am grateful also to the friends and collegues at the Multimedia Signal Processing Group of the Lisbon site of Instituto de Telecomunicações, and to the other friends I met in that city, who helped me in many aspects of my stay in Portugal, and made me feel at home.

As is tradition, I have left the most important and heart felt acknowledgements to the end. None of this would have been possible without the love and the help of my family who always encouraged me during the moments of difficulty and enjoyed with me my successes.

I want to reserve a particularly important thanks for my Mother Rosanna, who always believed in me and encouraged me throughout my years of my study, although unfortunately she is not able to see the result.