Performance and Design Guidelines for PPETP, a Peer-to-Peer Overlay Multicast Protocol for Multimedia Streaming

Riccardo Bernardini, Roberto Cesco Fabbro, Roberto Rinaldo
DIEGM – Università di Udine, Via delle Scienze 208, Udine, Italy
{riccardo.bernardini,roberto.cesco,roberto.rinaldo}@uniud.it

***Abstract*—One major issue in multimedia streaming over the Internet is the large bandwidth that is required to serve good quality content to a large audience. In this paper we describe PPETP, a peer-to-peer protocol for efficient multimedia streaming to large user communities. The performance of the protocol (such as the robustness of the protocol with respect to packet losses and churn) are quantitatively analyzed and guidelines for designing peer-to-peer streaming systems based on the described protocol are given.**

***Keywords*-Data transmission; multimedia streaming; overlay multicast; peer-to-peer network; push networks**

I. Introduction

A problem that is currently attracting attention in the research community is the problem of streaming live content to a large number of nodes. The main issue to be solved is due to the amount of upload bandwidth required to the server that, unless multicast is used, is equal to the bandwidth required by a single viewer (some Mb/s for DVD quality) multiplied by the number of viewers (that can be very large, for example, it is reported that in 2009 the average number of viewers per F1 race was approximately $6 \cdot 10^8$). Multicast could be a possible solution, but it has drawbacks too. For example, multicast across different Autonomous System (AS) has several issues, both technical and administrative ones.

An approach that recently attracted interest in the research community is the use of peer-to-peer (P2P) solutions as described in [2] to [15]. With the P2P approach each viewer re-sends the received data to other users, implementing what could be roughly defined as an overlay multicast protocol where each user is also a router. Ideally, if each user retransmitted the video to another user, the server would just need to "feed" a handful of nodes and the network would take care of itself.

Unfortunately, the application of the P2P paradigm to multimedia streaming has some difficulties. For example, depending on the media type and quality, residential users could have enough download bandwidth to receive the stream, but not enough upload bandwidth to retransmit it. This problem is known as the *asymmetric bandwidth* problem.

Another important issue with P2P networks of residential nodes is due to the *churn* of the network, that is, the "turbulence" induced by users joining and leaving the network at random. In particular, if a user suddenly leaves the network, other users could be left without data for a long time.

Moreover, P2P networks have several security issues [16]. Here we simply cite the *stream poisoning attack* where a node sends incorrect packets that cause an incorrect decoding and are propagated to the whole network by the P2P mechanism.

This article is the extension of [1] and describes the *Peer-to-Peer Epi-Transport Protocol* (PPETP), a peer-to-peer protocol developed at the University of Udine and hosted as part of the project *Corallo* on *SourceForge*. While the description given in [1] was more of a qualitative nature, describing the main feature of PPETP, without going into quantitative details, this paper aims to give a more analytical description of the feature of PPETP, with the objective of giving guidelines for designing networks based on PPETP. For the sake of completeness, in this paper we briefly summarize some results published elsewhere, taking care of marking explicitly the parts taken from other works.

This paper is organized as follows. Section III gives a qualitative overview of PPETP and introduces some jargon; Section IV introduces the concept of reduction procedure, a key concept in PPETP; Section V analyzes some features of PPETP that derives from the use of reduction functions; Section VI analyzes the packet loss probability experienced by the nodes in a PPETP network; Section VII gives some quantitative results about the robustness of PPETP against churn; Section VIII gives some guidelines for designing networks based on PPETP; Section IX presents the conclusions.

II. State of the art

The first P2P streaming networks had a tree structure, inspired by IP multicast. For example, ZIGZAG [17], built a multicast tree for media streaming at the application layer. This structure is, however, quite weak, mainly because, differently from IP-layer multicast, the P2P tree is built upon peers that may join and leave at any time. This is a serious issue, since a departing peer disconnects all its descendants from the source.

Multiple tree-based overlay architectures, such as SplitStream [10], CoopNet [18] and ChunkySpread [19], are proposed to mitigate the issues in single-tree architectures. Compared to architectures based on a single tree, architectures based on multiple trees are more resilient to peer departures

and failures. In addition, they can more efficiently use the uploading link capacity of each peer, since each peer works as an interior node in at least one tree. However, they achieve these benefits at the cost of more complicated architectures and media encoding methods.

Recently, many proposed P2P streaming networks, such as CoolStreaming [20], AnySee [21], PRIME [22], and DagStream [23] use mesh networks. In an *unstructured* mesh network (e.g., PRIME, CoolStreaming) a peer connects with a large number of randomly selected peers, with the purpose of providing more neighbors and more diverse paths. In a *structured* mesh network peers are typically grouped into clusters, in order to reduce the propagation delay of packets. However, such a locality-aware network may suffer from the shared bottleneck problem, where the media quality of all peers in a cluster strongly depends on the available bandwidth at a shared bottleneck. For these reasons, a locality-aware approach constructs a mesh with some special structure in order to achieve good network connectivity. Another approach used in some P2P streaming systems is the use of rateless codes. Examples of this approach are *rStream* [24] and *ToroVerde* [25].

Most of the currently available P2P streaming systems are mesh-based and employ ideas similar to the ones used in P2P file sharing systems such as BitTorrent. In this type of P2P streaming systems, the content is split into sections (called *chunks*, so this type of systems is sometimes referred to as *chunked* P2P systems). In a typical chunked system a node queries and requests them content chunks. A serious issue in chunk-based P2P systems is that they have very long start-up times due to the fact that in order to use a chunk-based system with live material, it is necessary to do some buffering.

III. Overview of PPETP

The goal of this section is to give a brief overview of the structure of PPETP and to introduce some PPETP jargon that will be used in the following. For the sake of brevity, many details will be omitted. A more detailed description can be found in the Internet Draft [26].

PPETP can be considered as a multicast overlay protocol based on a P2P approach that sends data and commands over a non necessarily reliable protocol (e.g., UDP). The type of multicast done over a PPETP network can be both Any Source Multicast (ASM) or Source Specific Multicast (SSM); in this paper we will consider, for the sake of concreteness, the SSM case only, the adaptations for the ASM being obvious. In the SSM case the origin of the content will be called *origin server*.

Since each node streams autonomously to other nodes, a PPETP network can be considered a *push* network. If node A receives data from node B, we will say that A is a *lower peer* of B and that B is an *upper peer* of A (therefore, data flows from top to bottom). PPETP does not mandate any specific network topology, the only constraint being that each node has a minimum number of upper peers.

Fig. 1 shows an example of a possible PPETP network for multimedia streaming with three upper peers per node. Each

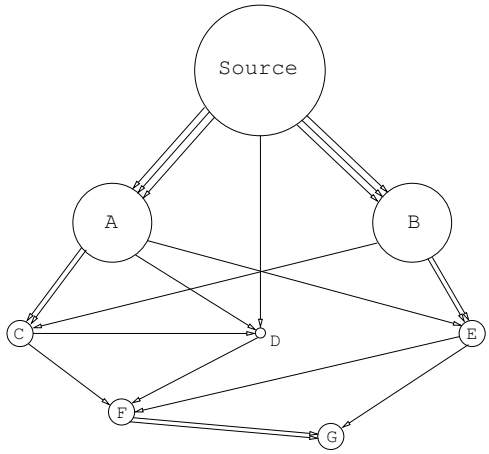


Figure 1. Example of a PPETP network for multimedia streaming.

arrow represents a stream, each circle represents a node and the node available upload bandwidth is represented by the circle size. For example in Fig. 1, node A (an upper peer of C, D and E) sends to C *two* different streams. Note also that the source "feeds" directly nodes A and B by sending them three different streams. Other examples of possible topologies for a PPETP network are shown in Fig. 2. Note that not only tree-structured networks are possible with PPETP.

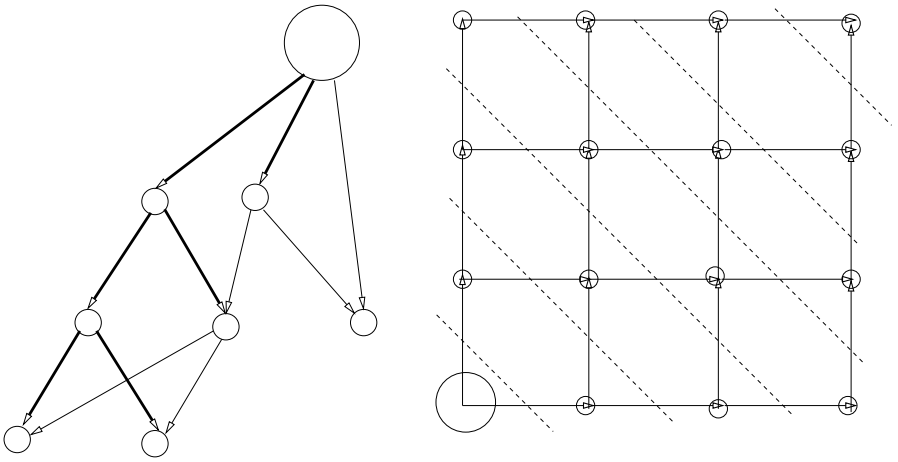
Remark III.1 (What PPETP is not)
A P2P streaming system is a complex piece of software that must take care of several things: transferring data, finding new peers, tracking content and so on. We would like to emphasize here that PPETP is designed to take care only of the efficient data distribution; other important aspects of the P2P streaming application (e.g., building the network) are demanded to extra-PPETP means. This is similar to what happens with TCP: the standard specifies how data is carried from a host to another, but does not specify, for example, how one host finds the other, this being handled by protocols such as DNS.

IV. Data reduction procedures

A key characteristic of PPETP is that, in order to solve the asymmetric bandwidth problem, every node does not send to its lower peers the whole content stream, but a *reduced stream* that requires less bandwidth. The reduced stream is obtained by processing each packet in the content stream with a suitable *reduction function*.

Informally, a reduction function is a function that maps the set of bit-strings (i.e., the set of packets) into itself, with the property that the result is shorter (actually, R times shorter) and that one can recover the original bit-string when at least R reduced versions if the original bit-string are known.

We will represent mathematically packets as elements of $\mathfrak{B} \stackrel{\text{def}}{=} \{0,1\}^*$, the set of all finite bit-strings. With this position, reduction functions are represented by functions mapping packets into packets, that is, $\mathfrak{B}$ in $\mathfrak{B}$. In the following will be convenient to have a notation that allows to represent compactly a vector of reduction functions.



(a) (b)

Figure 2. Examples of network topologies compatible with PPETP (a) parallel trees and (b) onion skin. The dashed lines mark the stratum boundaries

Notation 1. *Let* $\mathfrak{B} \stackrel{def}{=} \{0,1\}^*$ *be the set of all finite bit-string, let* J *be a finite set and let*

$$\mathfrak{R} = \{f_a : \mathfrak{B} \to \mathfrak{B}, a \in J\} \tag{1}$$

a set of functions mapping bit-strings in bit-strings and indexed by J*. For every n-ple of indexes* $\mathbf{a} = (a_1, a_2, \cdots, a_n) \in J^n$ *we will denote with* $g_{\mathbf{a}} : \mathfrak{B} \to \mathfrak{B}^n$ *the function*

$$g_{\mathbf{a}}(x) \stackrel{def}{=} [f_{a_1}(x), \ldots, f_{a_R}(x)] \tag{2}$$

The key property of reduction functions is what we call R-reconstruction property, that is, the possibility of recovering a packet when at least R different reduced versions of it are known. This idea is made precise in Definition 1.

Definition 1. *Set* $\mathfrak{R}$ *in (1) is said* to satisfy the R-reconstruction property *if for every* $\mathbf{a} \in J^R$ *the corresponding function* $g_{\mathbf{a}}$ *(defined as in (2)) is injective.*

We will say that the R-reconstruction property is satisfied tightly *by* $\mathfrak{R}$ *if for every* $\mathbf{a} \in J^{R-1}$ *the corresponding function* $g_{\mathbf{a}}$ *is* ***not*** *injective.*

If a set $\mathfrak{R}$ *satisfies tightly the R-reconstruction property we will also say that* $\mathfrak{R}$ is a set of reduction functions.

In the following the elements of J used to index the functions in $\mathfrak{R}$ will be called *reduction parameters*.

Remark IV.1

As anticipated, Definition 1 is a formal way to say that if (1) is a set of reduction functions, then it must be possible to recover $x \in \mathfrak{B}$ from the knowledge of any R-pla of reduced versions $f_{a_1}(x)$, …, $f_{a_R}(x)$. The condition of tight reconstruction helps in avoiding pathological cases that satisfy the R-reconstruction property but operates no reduction at all (e.g., when all the functions in $\mathfrak{R}$ are the identity function).

Since the idea of a set of reduction functions can seem a bit abstract and it can not be clear if a set of reduction functions exists at all, it is worth to give an example based on Reed-Solomon codes and used in PPETP with the name of *Vandermonde reduction profile* [26].

Example IV.1

Let $d > 0$ be an integer and let $\mathbb{F}_{2^d}$ denote the Galois field with 2^d elements. Galois field $\mathbb{F}_{2^d}$ will be used both as the reduction parameters set J and for computation.

The function $f_c : \mathfrak{B} \to \mathfrak{B}$ associated with reduction parameter $c \in \mathbb{F}_{2^d}$ is computed as follows. Let x be the argument of f_c, let

$$\mathbf{r}_c \stackrel{\text{def}}{=} \begin{bmatrix} 1 & c & c^2 & \cdots & c^{R-1} \end{bmatrix} \tag{3}$$

be the R-dimensional row vector in $\mathbb{F}_{2^d}^R$ whose components are powers of c and let $\mathbf{C}_x$ be the R-row matrix with entries in $\mathbb{F}_{2^d}$ by considering every d-ple of bits of x as an element of $\mathbb{F}_{2^d}$ (if the number of bits of the packet is not an integer multiple of dR, the packet is first suitably padded, see [26] for details). The value of $f_c(x)$ is

$$f_c(x) = \mathbf{r}_c \mathbf{C}_x \tag{4}$$

(Note that in (4) we did a notational abuse, since the value of $f_c(x)$ should be a bit-string, while the right hand side of (4) is a vector of elements of $\mathbb{F}_{2^d}$.)

In order to see that the set of functions f_c is actually a set of reduction functions with reduction factor R, observe that from the knowledge of $f_{c_1}(x), \ldots, f_{c_R}(x)$ one can recover $\mathbf{C}$ by solving the linear system

$$\begin{bmatrix} f_{c_1}(x) \\ f_{c_2}(x) \\ \vdots \\ f_{c_R}(x) \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{c_1} \\ \mathbf{r}_{c_2} \\ \vdots \\ \mathbf{r}_{c_R} \end{bmatrix} \mathbf{C}_x = \underbrace{\begin{bmatrix} 1 & c_1 & \cdots & c_1^{R-1} \\ 1 & c_2 & \cdots & c_2^{R-1} \\ \vdots & \vdots & & \vdots \\ 1 & c_R & \cdots & c_R^{R-1} \end{bmatrix}}_{\mathbf{R}} \mathbf{C}_x \tag{5}$$

Since matrix $\mathbf{R}$ in (5) is a Vandermonde matrix, it is invertible (and (5) has a solution) as soon as all the c_k are different.

Remark IV.2

Although the approach in Example IV.1 is well known, many other sets of reduction functions can be constructed; see, for example, [27].

By exploiting the idea of reduction functions, nodes of a PPETP network propagate the streamed data as follows

- At start-up
  1) Each node chooses one or more reduction parameters a_1, a_2, …. Although the parameter(s) can

be imposed by an external network coordinator, if the parameter space is large enough, the nodes can choose them at random, simplifying the network management. See Section V-B.

2) Contact $N_{up} \geq R$ upper peers. Each upper peer will communicate to the node its reduction parameter before starting streaming.

- For every content packet
  1) Wait for at least R different reduced packets
  2) After receiving at least R reduced packets, recover the content packet
  3) Move the content packet to the application level
  4) Reduce the content packet using the chosen reduction parameters $a_1, a_2, \ldots$.
  5) Send the computed reduced packets to your lower peers

Note that if, because of packet losses, the node receives less than R reduced versions of the content packet, the node can still help in propagating the information by forwarding to its lower peers the reduced data received from the upper peers. We call this (almost obvious) strategy *fragment propagation* and it will be shown in the following that, despite of its simplicity, it is important for system performance.

Remark IV.3

The strategy of fragment propagation can help in solving a problem that is intrinsic to the P2P network of residential nodes. If a residential node has the upload bandwidth smaller than the content bandwidth, it introduces a "bandwidth debt" since it cannot compensate the consumed download bandwidth with an equal upload bandwidth. Such a debt must be covered by other nodes such as super-nodes or by the origin server. Since the bandwidth debt can be expected to grow linearly with the number of residential nodes, this problem can challenge the scalability of the P2P network.

The use of reduction functions and fragment propagation can help in counteracting this problem. A residential node can act as a "repeater" (maybe in exchange of some improved service) by simply joining the network and contacting only one upper peer, but accepting $N_{low} > 1$ lower peers. Automatically, because of the fragment propagation policy, it will forward to its lower peers the packets received from the upper peer. The overall effect is a "bandwidth gain" equal to $N_{low} - 1 > 0$ reduced streams that compensates the debt of other nodes.

A. Data puncturing

In the case of high quality content (that requires a large bandwidth) and a network with low upload bandwidth nodes, it could happen that the required reduction factor R is too large (the drawbacks of a too large reduction factor will become clear in the following). In this case PPETP can reduce the upload bandwidth by *puncturing* the stream of fragments. Puncturing can be both *probabilistic* or *deterministic*. In the former case, the packets to be sent are chosen randomly with a given probability, in the latter case the packets are chosen according to a pattern that is periodically repeated (e.g., send only the even packets). For example, Fig. 3a shows a node with five upper peers, where two peers (nodes C and D) apply a 1:2 puncturing to the data stream, node C sending only even packets and D only odd ones. It is clear that the scheme of Fig. 3a is approximately equivalent to the scheme of Fig. 3b, where the two puncturing nodes are "merged" in a "virtual" no puncturing node.

V. Properties of Data Reduction

In this section we discuss few interesting properties due to the use of data reduction schemes. It is interesting to observe that the properties discussed in this and the following sections do not depend on the actual functions f_a, but only on their property of being reduction functions. Therefore, the properties discussed here hold not only for the Vandermonde scheme of Example IV.1, but also for any other reduction scheme that enjoys the R-reduction property.

A. Solution to the asymmetric bandwidth problem

This property is almost obvious, but it is included here for the sake of completeness. Since the bit-string associated with the size of the reduced packet is R smaller than the size of the content packet, the bandwidth required by the reduced stream is R times smaller. By choosing R large enough, even the nodes with small upload bandwidth can contribute to propagating the content.

B. Distributed assignment of the reduction function

A first interesting property is that if the set of reduction parameters J is large enough, each node can choose its parameter at random, since the probability of having two nodes with the same reduction parameter is negligible. This simplifies the assignation of the reduction parameters to the nodes, since a central authority is not required.

Remark V.1

Note that there is no computational overhead in choosing $|J|$ large, since J represents the "pool" from which reduction functions are chosen, that can be much larger than the number of generated reduction packets. For example, in the PPETP specs [26] $|J| = 2^{32}$ although R can be expected to be at most ≈ 20.

In order to be more quantitative, let $S = |J|$ be the cardinality of the reduction parameter set J, let N_{up} be the number of upper peers of a given node and let a_k be the reduction parameter of the k-th upper peer, $k = 1, \ldots, N_{up}$. The node is able to recover the content stream if and only if there are least R different values of a_k. Fig. 4 shows the probability P_{fail} that this does not happen as a function of S, the reduction factor R and the ratio $\rho = N_{up}/R$ (interpretable as a redundancy factor). It is clear from Fig. 4 that one can achieve negligible P_{fail} by using J of reasonable size and small redundancy factors. The curves in Fig. 4 have been obtained by means of the numerical procedure described in Appendix A where it is also shown that P_{fail} goes to zero with N_{up} as

$$[(R-1)/S]^{N_{up}} = [(R-1)/S]^{\rho R} \quad (6)$$

By means of standard analysis techniques, it is possible to show that (6), as a function of R, has a single minimum at $R \approx 1 + S/e$. Since S is typically very large (for example, $S = 2^{16}$ if $\mathbb{F}_{2^{16}}$ is used in the reduction scheme of Example IV.1),

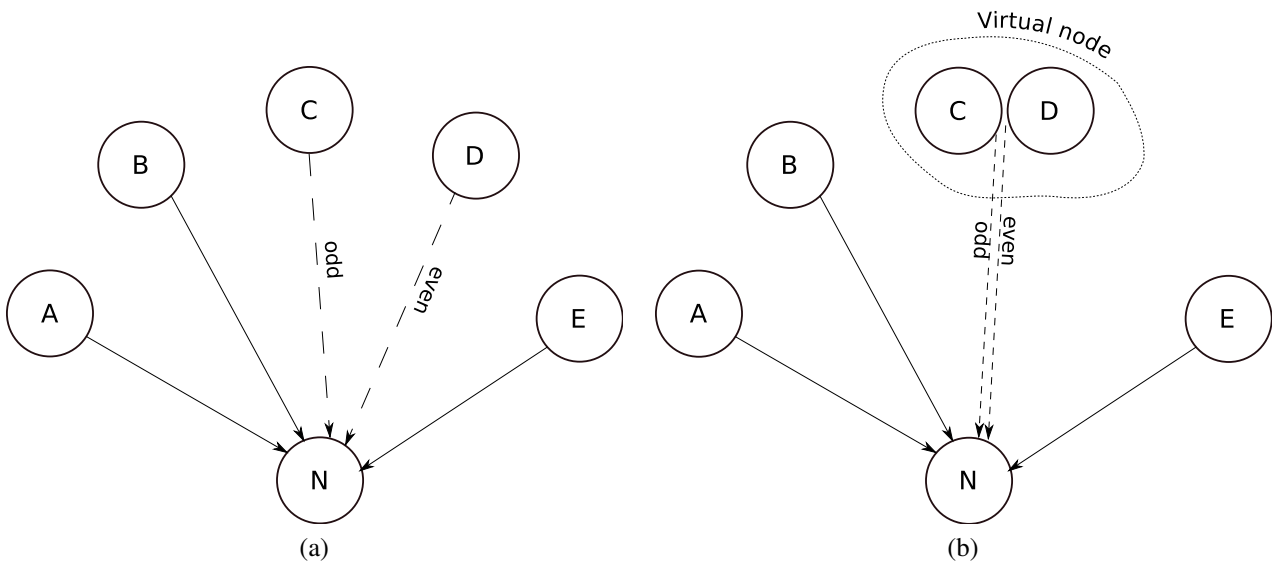


Figure 3. (a) Node N has five upper peers, with two upper peers (C and D) applying a puncturing 1:2. (b) Network equivalent to the network in (a) with a "virtual" upper peer obtained by merging nodes C and D.

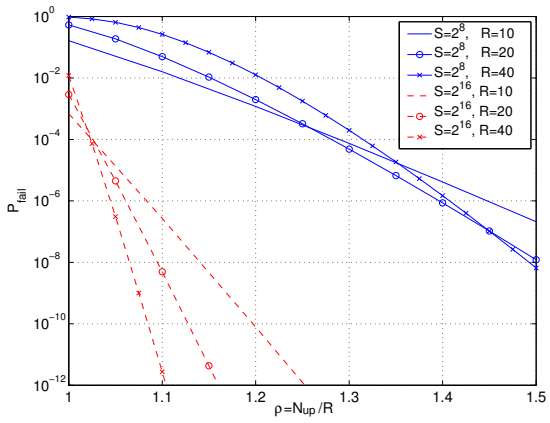


Figure 4. Probability P_{fail} of having less than R different reduction parameters out of N_{up} vs. redundancy ratio $\rho = N_{\text{up}}/R$ for different values of $S = |J|$ and R.

it follows that (6) decreases monotonically with R in every case of practical interest. Moreover, since the minimum of (6) is very low when S is large, it follows that, fixed ρ, one can make P_{fail} as small as desired by taking R large enough.

C. Robustness with respect to packet loss

The scheme is inherently robust with respect to packet losses, typically due to peer departures and network congestion. Actually, a node that contacts $N_{\text{up}} > R$ upper peers will be able to recover the transmitted data as long as not more than $N_{\text{up}} - R$ packets are lost. The effect of packet losses is discussed in more detail in Section VI.

D. Robustness with respect to churn

An important problem in P2P streaming network is that a node can leave at anytime, leaving its lower peers without data. The fact that in a network made of residential users one can expect a high *churn* is one of the reasons that support the use of mesh-based chunky solutions. However, in PPETP the same redundancy that protects against packet losses protects also against the effect of churn. This is discussed in more detail in Section VII.

E. Robustness to stream poisoning

As said above, a possible attack is the injection of "garbage packets." The use of reduction functions offers a simple way to counteract such a threat. Actually, it suffices to contact $N_{\text{up}} > R$ upper peers, use R reduced versions to recover the original data and check that the result is coherent with the remaining reduced packets. More precisely, let u_k denote the packet received by the k-th upper peer, let a_k denote the corresponding reduction parameter and let $\mathbf{a} = [a_1, \ldots, a_R]$ and suppose that at most $N_{\text{up}} - R - 1$ packets u_k can be corrupted.

In order to recover x safely, the node chooses R reduced packets u_{k_1}, ..., u_{k_R}, recovers the content packet as $x = g_{\mathbf{a}}^{-1}(u_{k_1}, \ldots, u_{k_R})$ and checks the result by verifying that the following equalities hold

$$u_{k_\ell} = f_{\alpha_{k_\ell}}(x), \qquad \ell = R+1, \ldots, N_{\text{up}} \tag{7}$$

The following cases may happen

1) All the equalities (7) are verified. In this case x is correctly recovered and every peer sent us a correct packet.
2) Some of the equalities (7) are verified, but not all. In this case x is still correctly recovered, but the peers corresponding to the non verified equalities sent us a corrupted packet.
3) All the equalities (7) are not satisfied. Since we supposed that at most $N_{\text{up}} - R - 1$ packets u_k can be corrupted, this can happen only if we used a corrupted packet in recovering x. In this case we can choose a different set

of u_k and try to recover x again. If only one corrupted packet is present, the expected number of trials before x is recovered is $N_{\text{up}}/(N_{\text{up}}-R)$. If the recovering of x has already been attempted too many times, one can declare the packet lost and let the application conceal the loss.

The procedure described above can be considered as a generalization of the use of error correcting codes. However, in this case data are not corrupted by a noisy channel, but by a malicious attacker that could, in principle, send carefully crafted data that cause the node to recover garbage data that nevertheless passes the test above. Therefore, the question is: can an attacker craft a corrupted packet $\hat{u}_k$ that produces a wrong packet $\hat{x} \neq x$ that satisfies the test above? What about a coordinated attack by A peers? We are going to show that if $N_{\text{up}} \geq R+A$, the system is immune from a coordinated attack by A peers.

To be more precise, let x be the original content packet and suppose a given node receives data from $N_{\text{up}} > R$ peers. Let $u_k = f_{a_k}(x)$ be the reduced packet that the node *should* receive from peer k and let $\hat{u}_k$ be the actual received packet. Since we are supposing that no more than A peers will try a coordinated attack, we know that there are at most A values of k such that $\hat{u}_k \neq u_k$.

The packet recovered by the node is $\hat{x} = g_{\mathbf{a}}(\hat{u}_1,\ldots,\hat{u}_R)$ and the node accepts it if all the following equalities hold

$$f_{a_\ell}(\hat{x}) = \hat{u}_\ell, \qquad \ell = R+1,\ldots,N_{\text{up}}. \tag{8}$$

We can say that *the attack succeeds if $\hat{x} \neq x$ and the node accepts $\hat{x}$*. The following theorem shows that a coordinated attack by A peers fails if $N_{\text{up}} \geq A+R$.

Theorem 1. *Let $x \in \mathfrak{B}$, let $a_1,\ldots,a_{R+A} \in J$, and let $u_k = f_{a_k}(x)$, $k = 1,\ldots,R+A$. Let $\hat{u}_k \in \mathfrak{B}$, $k = 1,\ldots,R+A$ be such that $\hat{u}_k \neq u_k$ for at most A values of k. Let $\mathbf{a} = [a_1,\ldots,a_R]$ and define*

$$\hat{x} \stackrel{def}{=} g_{\mathbf{a}}^{-1}(\hat{u}_1,\ldots,\hat{u}_R) \tag{9}$$

The following equalities hold

$$f_{a_\ell}(\hat{x}) = \hat{u}_\ell, \qquad \ell = R+1,\ldots,R+A \tag{10}$$

if and only if $x = \hat{x}$.

Proof: As a first step, we show that $\hat{x}$ satisfies $f_{a_k}(\hat{x}) = \hat{u}_k$ for all the $k = 1,\ldots,R+A$. Indeed, such an equality is satisfied for $k \leq R$ because of definition (9) and it is satisfied for $k > R$ because of (10). By hypothesis, there are at least R integers $n_1, n_2, \ldots, n_R \in \{1,\ldots,R+A\}$ such that $u_{n_k} = \hat{u}_{n_k}$. Let $\mathbf{a} = [a_{n_1},\ldots,a_{n_R}]$ and observe that

$$\begin{aligned} g_{\mathbf{a}}(\hat{x}) &= [f_{a_{n_1}}(\hat{x}),\ldots,f_{a_{n_R}}(\hat{x})] \\ &= [\hat{u}_{n_1},\ldots,\hat{u}_{n_R}] = [u_{n_1},\ldots,u_{n_R}] \\ &= [f_{a_{n_1}}(x),\ldots,f_{a_{n_R}}(x)] = g_{\mathbf{a}}(x) \end{aligned} \tag{11}$$

By Definition 1, (11) holds only if $x = \hat{x}$. ■

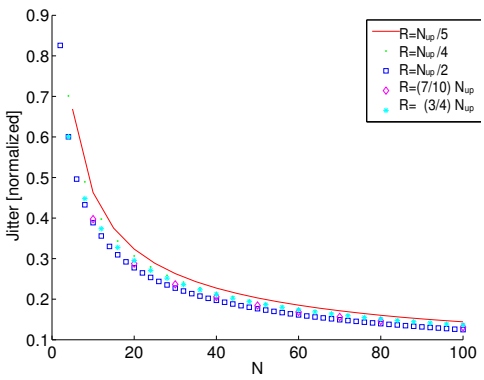


Figure 5. Jitter as a function of N (peer delays distributed as Gaussian with unit variance)

F. Jitter reduction

A nice side effect of the use of network coding in PPETP is, as reported in [28], the reduction of the jitter observed by the node. Intuitively, this happens because the time when a content packet is recovered is the time necessary for the arrival of the R fastest packets out of N_{up}. Fig. 5, taken from [28], shows the theoretical prediction of the jitter (i.e., the standard deviation of the reconstruction time), as a function of R and N_{up}, when the delays are Gaussian with variance σ^2. The values on the vertical axis are measured in units of σ. Note that the jitter decays as $1/\sqrt{N_{\text{up}}}$ [29]. This behavior was also verified experimentally [28].

G. Computational cost

It is convenient to analyze briefly the cost of the computation due to the reconstruction and reduction with the Vandermonde profile, in order to get an estimate of how that cost depends on the design parameters (R, d and N_{up}).

Let Q be the size (in bits) of a content packet. If we work with the Galois field $\mathbb{F}_{2^d}$, the matrix corresponding to the packet will have Q/d entries organized in R rows and $Q/(dR)$ columns (for the sake of notational simplicity, we are supposing that Q is an integer multiple of dR). Let $C_+(d)$ and $C_\times(d)$ be the "cost" associated with, respectively, a sum and a product in $\mathbb{F}_{2^d}$. As the unit of measure of the cost, we will take the time required to do a 32-bit XOR that corresponds to a sum in $\mathbb{F}_{2^{32}}$ and that is implemented with a single instruction on modern microprocessors.

The reconstruction step requires a product between a $R \times R$ matrix (the inverse of the Vandermonde matrix) and the $R \times Q/(dR)$ matrix obtained by stacking the row vectors corresponding to the reduced packets. Such a matrix product requires

$$R \cdot R \cdot Q/(dR) = RQ/d \qquad \text{products} \tag{12a}$$

$$R \cdot (R-1) \cdot Q/(dR) \approx RQ/d \qquad \text{sums} \tag{12b}$$

The reduction step requires the product of the $1 \times R$ reduction vector by the $R \times Q/(dR)$ matrix corresponding to the content

packet. This requires

$$1 \cdot R \cdot Q/(dR) = Q/d \qquad \text{products} \qquad (13a)$$

$$1 \cdot (R-1) \cdot Q/(dR) \approx Q/d \qquad \text{sums} \qquad (13b)$$

Therefore, the overall computational cost per a Q-bit packet is

$$\frac{C_+(d)+C_\times(d)}{d}(1+R)Q = \overline{C}(1+R)Q \qquad (14)$$

where $\overline{C} = (C_+(d)+C_\times(d))/d$ can be interpreted as a "computational cost per bit" due to the operations on the Galois field. If B is the content bit-rate in bit/s, we need to process a packet every Q/B seconds, so that we have a computational load equivalent to

$$\frac{\overline{C}(1+R)Q}{Q/B} = \overline{C}(1+R)B \qquad \text{32-bit XOR/s} \qquad (15)$$

Note that the computational cost grows linearly with the reduction factor.

1) Cost of the operations in $\mathbb{F}_{2^d}$: It is clear that the term $\overline{C} = (C_+(d)+C_\times(d))/d$ depends on the algorithm used for implementing the Galois operations and on the specific architecture. Nevertheless, in order to have a grasp on the dependence of this term on d, we carried out few experiments.

We considered the following possible implementations for a product in $\mathbb{F}_{2^d}$

Long product
: The product in $\mathbb{F}_{2^d}$ is done with an algorithm similar to the integer product algorithm.

Logarithm table
: If 2^d is not too large (say, up to $d = 16$), one can exploit the possibility of defining a logarithm in $\mathbb{F}_{2^d}$ and do the product using a "logarithm table."

Pythagorean table
: If d is quite small (say, up to $d = 8$), one can do the product using a Pythagorean table that stores the product for every possible pair of values.

Kronecker via Pythagorean table
: If d is very small, (e.g., $d = 4$) one can use the Pythagorean table approach to compute more than one product at once. For example, if $d = 4$, $\mathbf{a} = [a_1, a_2] \in \mathbb{F}_{2^4}^2$ and $\mathbf{b} = [b_1, b_2] \in \mathbb{F}_{2^4}^2$, one can compute the Kronecker product $\mathbf{a} \otimes \mathbf{b} = [a_1 b_1, a_2 b_1, a_1 b_2, a_2 b_2]$ by concatenating a_1, a_2, b_1 and b_2 and using the resulting 16-bit index to access a look-up table with the entries of $\mathbf{a} \otimes \mathbf{b}$. The cost of this approach is comparable with the cost of the Pythagorean table approach, but it allows to compute four products at once. An example of this approach can be seen in Fig. 10.a (in assembler) and in Fig. 11 (in C) in Appendix A.

We implemented (in Assembler, in order to avoid the influence of compiler optimizations) the product algorithms described above for $d = 4, 8, 16$ and 32. The source code (with the syntax of the GNU assembler [30] of the implemented procedures is reported in Appendix A, Fig. 10. The time required by those procedure has been measured by means of the `RDTSC` (ReaD Time Stamp Counter), an instruction of x86 processors that allows to obtain the value of the Time Stamp Counter, a 64-bit register increased at each clock cycle. [31]. Note that the programs in Fig. 10 do not include, for the sake of space, side-code such as parameter handling code. The complete set of sources is available, upon request, from the author.

Table I
APPROXIMATE RELATIVE COMPUTATIONAL COST $C_\times(d)$ OF THE PRODUCTS AND COST PER BIT $\overline{C}$ IN DIFFERENT GALOIS FIELD. THE UNITARY RELATIVE COST IS A 32-BIT XOR.

Field	$C_\times(d)$	Cost per bit $\overline{C}$	Memory	Notes
$\mathbb{F}_{2^4}$	0.5	1.5/4 = 0.375	128 K	Kronecker
$\mathbb{F}_{2^8}$	1	2/8 = 0.250	64 K	Pythagorean table
$\mathbb{F}_{2^{16}}$	3.5	4.5/16 = 0.281	256 K	Logarithm table
$\mathbb{F}_{2^{32}}$	20	21/32 = 0.656	0	Long product

The results of such measurements can be seen in Table I that shows the relative complexity of the product in $\mathbb{F}_{2^d}$ and the corresponding "cost per bit" $\overline{C}$ for some values of d, where the computational cost is measured, as anticipated, relatively to the computational cost of a 32-bit XOR. It is worth observing that the overall cost per bit does not change much with the size of the Galois field, with the most expensive field being $\mathbb{F}_{2^{32}}$.

Remark V.2
It is worth observing that the algorithms chosen for the experiments and the representation used for the elements of $\mathbb{F}_{2^d}$ are not the only possible. The choice of the "best" representation of elements of $\mathbb{F}_{2^d}$ and of the "best" algorithms can be done *only* once the architecture has been chosen since, especially with procedures as short as the ones presented here, details such as the internal structure of the processor, memory alignment, cache, and maybe others can play a non-negligible role. Therefore, the complexity figures given in this section should be taken only as *planning figures*.

H. Information obfuscation

The reduction procedure described in Example IV.1 has some similarity with the secret sharing technique of Shamir [32]. This suggests that one could use it to add some protection to the transmitted content. In order to simplify the discussion, we need a new definition.

Definition 2. *We will say that a reduction scheme with reduction factor R* achieves *k*-secrecy *if, given*

- *A positive integer K*
- *Any k-ple of K-dimensional row vectors (representing reduced packets) $\mathbf{u}_{c_1}, \ldots, \mathbf{u}_{c_k} \in \mathbb{F}_{2^d}^K$*
- *Any content packet $\mathbf{C}$ with R rows and K columns,*

it is possible to find $R-k$ reduced packets $\mathbf{u}_{c_{k+1}}$ $\mathbf{u}_{c_R} \in \mathbb{F}_{2^d}^K$ so that the content packet recovered from the whole set of reduced packets $\mathbf{u}_{c_1}, \ldots, \mathbf{u}_{c_R}$ is $\mathbf{C}$.

Definition 2 formalizes the idea that, if k-secrecy is achieved, an opponent that gets to know no more than k reduced packets, cannot deduce anything about the original content packet since any content packet can give rise to ("is

compatible with") the sequence of eavesdropped packets $\mathbf{u}_{c_1}$, ..., $\mathbf{u}_{c_k}$. Note that the reduction scheme in Example IV.1 does not achieve even 1-secrecy since, given any reduced packet, there are many content packets that are not compatible with it.

We want to show how the scheme in Example IV.1 can be modified to obtain 1-secrecy. Let K be the number of columns of $\mathbf{C}$ and let $\eta^t \in \mathbb{F}_{2^d}^K$ a random row vector whose entries are iid and uniformly distributed over $\mathbb{F}_{2^d}$. Use η to extend $\mathbf{C}$ to obtain

$$\widehat{\mathbf{C}} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{C} \\ \eta \end{bmatrix} \tag{16}$$

Now reduce matrix (16) using, of course, a reduction vector with $R+1$ columns, that is,

$$\hat{\mathbf{u}} = [1, c, \ldots, c^R]\widehat{\mathbf{C}} = \mathbf{r}_c\mathbf{C} + c^R\eta \tag{17}$$

Suppose now that an eavesdropper gets to know the reduced version $\hat{\mathbf{u}}$; we claim that the eavesdropper cannot deduce anything about $\mathbf{C}$. The reason is that for every choice of $\mathbf{C}$ one can find η such that (17) is satisfied, indeed

$$\eta = c^{-R}(\hat{\mathbf{u}} - \mathbf{r}_c\mathbf{C}) \tag{18}$$

where c^{-R} makes sense since c is a non-null element of $\mathbb{F}_{2^d}$. We have achieved 1-secrecy. It is easy to prove that k-secrecy can be achieved by extending $\mathbf{C}$ with a k-row random matrix.

Remark V.3

Although this technique seems to be specific for the Vandermonde reduction procedure, it can be extended to a more general case, as shown in [27], where it is also shown that a slightly stronger form of k-secrecy is achieved, that is, that the mutual information [33] between the content packet $\mathbf{C}$ and the reduced value $\hat{\mathbf{u}}$ is zero.

The advantage of this technique with respect to usual cryptography is that it does not require any key distribution, the drawbacks are an increased bandwidth (the reduced packets have the same dimension, but now a node must receive at least $R+k$ reduced packets instead of R) and the fact that an adversary that can get all the needed reduced packets can recover the content. If those drawbacks are compensated by the simplification due to the fact that no key distribution is necessary, depends on the applicative context.

VI. Packet loss probability

The current version of PPETP runs over UDP that, as well known, is an unreliable protocol. This means that a fragment sent to a lower peer could not reach its destination. It is clear that the probability that a given peer reconstructs a packet is a complex function of the packet loss probability and network structure. It is also clear that it is important to have an estimate, as precise as possible, of the overall packet loss probability experienced by a node. In this section we present some preliminary results about this topic.

A. Network model

A PPETP network can be represented by a Direct Acyclic Graph (DAG) where edges link each node to its lower peers, and where the server(s) is (are), clearly, the node(s) that do not have any upper peer. For the sake of notational simplicity, we will suppose that every link is an erasure channel that drops packets with probability P_ℓ.

We associate with each node n of the network the random variable W_n defined by the following experiment. We let the server(s) send to the network a single content packet, and we let $W_n \in \{0, 1, \ldots, N_{\text{up}}\}$ be the number of fragments received by node n. From the knowledge of the statistical properties of W_n, it is possible to determine several values of interest. For example, the packet loss probability P_{eq} seen by the application can be computed as $P_{\text{eq}} = P[W_n < R]$.

As explained in paragraph IV, 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 $T = R$ otherwise. If node n received at least T reduced packets (i.e., if $W_n \geq T$) we will say that the node is *active* or in *firing state*. We will define the random variable F_n to be equal to 1 if node n is in firing state and 0 otherwise.

1) Network topology: A 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 it is not easily captured by a small set of parameters. In order to simplify the study, it is convenient to put some constraint on the topology.

A useful constraint that nevertheless is 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 (since the network is a DAG there is a finite number of 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 $D(n)$ will be called the *depth* of node n, and difference $D(n) - d(n)$ will be called the *spread* of n. The network will be 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 this type of networks will be the natural outcome of the tentative of maximizing locality.

In this paper, we consider a special case of limited spread networks, namely, *stratified* networks; we use the term *stratified* to avoid confusion with the term *layered* possibly used in other contexts. In a stratified network, the nodes can be partitioned into sets (*strata*) $\mathfrak{L}_K$, $K \in \mathbb{N}$, such that all the upper peers of a node in $\mathfrak{L}_K$ belong to $\mathfrak{L}_{K-1}$. It is easy to verify that a network is stratified if and only if it has 0-limited spread and that the stratum index coincides with the node depth. Fig. 2 shows few examples of stratified networks, namely a tree network, a network made of parallel trees and an "onion skin" network. (Onion skin networks are interesting because the ratio of non-streaming nodes goes to zero when the network size goes to infinity.)

B. Notation

In this section we will consider Markov chains with a finite alphabet. We will use $\rightarrow$ to denote a one-step reachability relation, that is, we will write $a \rightarrow b$ if the chain can transition from a to b in one step. We will use $a \rightarrow^n b$ 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, we will use the shorthand $P(a \rightarrow b_1 \rightarrow b_2 \rightarrow \cdots b_N)$ to denote $P[s_{n+N} = b_N, \ldots, s_{n+1} = b_1 | s_n = a]$. Note that this notation factorizes, that is, $P(a \rightarrow b \rightarrow c) = P(a \rightarrow b)P(b \rightarrow c)$.

1) Notation for stratified networks: We will denote with L_K the number of nodes in stratum K. The n-th node in stratum K, $n = 0, \ldots, L_K - 1$, will be named as (K,n). The set of upper peers of (K,n) will be 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.

We will collect all the random variables $W_{K,n}$ and $F_{K,n}$, relative to nodes of stratum K, in two vectors $\mathbf{W}_K$ and $\mathbf{F}_K$ defined as

$$[\mathbf{W}_K]_n = W_{K,n} \qquad ; \qquad [\mathbf{F}_K]_n = F_{K,n} \tag{19}$$

Note that $\mathbf{F}_K \in \{0,1\}^{L_K}$. It will prove useful to have a special notation for some states in $\{0,1\}^{L_K}$. More precisely, we will define the *empty state* as $\phi = [0,0,\ldots,0]$ (no node in active state), the *full state* as $\Omega = [1,1,\ldots,1]$ (every node in active state) 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).

C. Equivalent loss probability

Consider a node (K,n) in stratum K and consider the following experiment: the origin server sends a content packet over the network and we check if node (K,n) recovers the content packet or not. Our goal is to obtain a bound to the equivalent loss probability P_{eq}, that is, the probability that the node does not recover the packet. It will be more convenient, from a notation point of view, to bound the probability of the complementary event $1 - P_{\text{eq}}$.

Property 1. *The following bound holds*

$$1 - P_{eq} \geq \eta \lambda^{\sum_{n=1}^{K} L_n} \tag{20}$$

where

$$\eta = P[\mathscr{B}(N_{up}, P_T) \geq R] \tag{21a}$$

$$\lambda = P[\mathscr{B}(N_{up}, P_T) \geq T] \tag{21b}$$

and $\mathscr{B}(N_{up}, P_T)$ in (21) is a binomial random variable with N_{up} trials and success probability P_T.

The proof of Property 1 is given in Appendix B.

Remark VI.1

Note that if fragment propagation is employed, $T = 1$ and (21b) can be written as

$$\lambda = 1 - P_\ell^{N_{\text{up}}} \tag{22}$$

Bound (20) decays exponentially with the number of peers in the network ($\sum_{n=1}^{K-1} L_n$ is the number of peers in the strata above stratum K), so it would seem not a very good bound. However, note that since the empty state ϕ is absorbing (that is, when a stratum reaches ϕ every successive strata will remain in ϕ), a well-known results on Markov chains implies that the probability of the empty state goes to 1 when the number of strata goes to infinity, so that the probability of reconstruction must converge to zero. Therefore, any lower bound of such a probability must converge to zero, too.

It is worth considering a simple numerical example, in order to understand better bound (20) and the difference between using or not fragment propagation. Suppose, for the sake of this example, that $P_\ell = 0.1$, that every node has $N_{\text{up}} = 10$ upper peers, that the reduction factor is $R = 6$ and that every stratum has $L = 100$ nodes.

If fragment propagation is employed, according to (22)

$$\lambda = 1 - P_\ell^{N_{\text{up}}} = 1 - 10^{-10} \tag{23}$$

Suppose we want to find $M = \sum_{n=1}^{K-1} L_n$ such that term λ^M becomes equal to 0.999. It is

$$M = \frac{\log_{10} 0.999}{\log_{10}(1 - 10^{-10})} = \frac{-4 \cdot 10^{-4}}{-4 \cdot 10^{-11}} = 10^7 \tag{24}$$

that corresponds to 10^5 strata if every stratum has 100 nodes. That is, although the bound (20) goes to zero when the network size goes to infinity, the decay is slow enough to be negligible for all but very large networks.

If no fragment propagation is employed, the value of λ is

$$\lambda = P[\mathscr{B}(10, 0.9) \geq 5] \approx 1 - 1.410^{-4} \tag{25}$$

Note that without fragment propagation, the term λ^M becomes smaller than 0.999 already with $M = 7$. Although (20) is only a lower bound, it suggests that convergence to the empty state can be very fast if fragment propagation is not used.

It is also worth observing that while the value of λ without fragment propagation depends on the redundancy $\rho = N_{\text{up}}/R$ (we had to use $\rho = 2$ in (25) in order to get a fairly large value for λ), the value of λ without fragment propagation *depends only on N_{up}* and we can obtain values of λ very close to 1 even with ρ small.

D. Repeated fragments

Note that in the proof of Property 1 it was implicitly assumed that the fragments received by the node were all different and one could wonder how much this hypothesis is true in a real case. This section is devoted to the discussion of this hypothesis.

First observe that, according to the results of Section V-B, we can safely assume that the reduction parameters chosen by the upper peers of a given node are different one another as soon as the number $|J|$ of reduction parameters is large enough. Moreover, if the network is not too large we can assume that the reduction parameters chosen by all the the *ancestors* of a given node are different.

Therefore, if the number of reduction parameters is sufficiently large with respect to the network size, if a node receives the same fragment twice, both fragments must have been originated by a single node. This happens, for example,

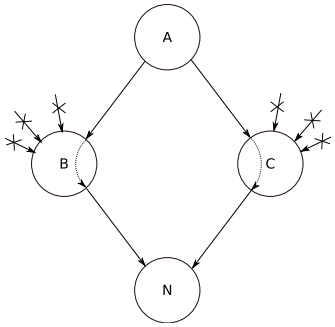


Figure 6. Example of a duplicated fragment event: both nodes B and C receive only the fragment from A and forward it to node N.

in the case of the *diamond* shown in Fig. 6 where node A sends a fragment to nodes B and C, both upper peers of N. If both B and C receives less than R fragments, it is possible that they both send to N the fragment received from A. Note that if N_{up} and R are suitably chosen, the probability that this happens is very small since it is necessary (but not sufficient) that both B and C cannot recover the corresponding content packet.

Since the problem of having an estimate of the event of duplicated fragment is still open, we decided to validate the effect of the hypothesis of no duplicated fragment by carrying out some simulations reported in Section VI-E.

E. Experimental results

We carried out some simulations in order to verify the theoretical results above and to asset the importance of the hypothesis of no duplicated fragment in a real case. For every choice of parameters P_ℓ, R, N_{up} we generated 20 random networks with 15 nodes per stratum, each node with N_{up} upper peers. Over each network we sent 1000 content packets and measured the probability (averaged over all the networks) that a node of a given stratum is able to recover the content packet. We carried out the simulations both with and without the fragment propagation policy. For each fragment we tracked its reduction parameter, therefore taking into account in the simulation the event of duplicated fragments. When a node is not able to recover the content packet, it selects one of the received fragments at random and forwards that to the lower peers.

According to the theoretical results described in Section VI-C we make the following predictions

- In the case *with* fragment propagation, with $P_\ell^{N_{\text{up}}}$ small, we expect P_{eq} approximately equal to the loss probability experienced when protecting packets sent over a channel with erasure probability P_ℓ using a (N_{up}, R) code, practically independent on the stratum number.
- In the case *without* fragment propagation we expect a P_{eq} that converges very rapidly to 1.

Fig. 7 shows, on a logarithmic scale, the probability of packet recovery $1-P_{\text{eq}}$ as a function of the stratum number for the cases $P_\ell = 0.2$, $N_{\text{up}} = 13$, redundancy factor R equal to 7 (first row), 8 (second row) and 9 (third row), with and without fragment propagation (left and right hand column, respectively). Observe that the probability remains approximately constant for all the cases with fragment propagation, while it decreases rapidly when fragment propagation is not employed. Note the reduced stratum range for figures Fig. 7b2 and Fig. 7c2; if we used the same range of the other figures, the curve would have looked like a vertical line. Note also that although in the case of Fig. 7a2 the probability does not decay as fast as in the other two figures of the same column, the decay is perceptible, while it is practically invisible in the three figures of the first column, relative to the fragment propagation case.

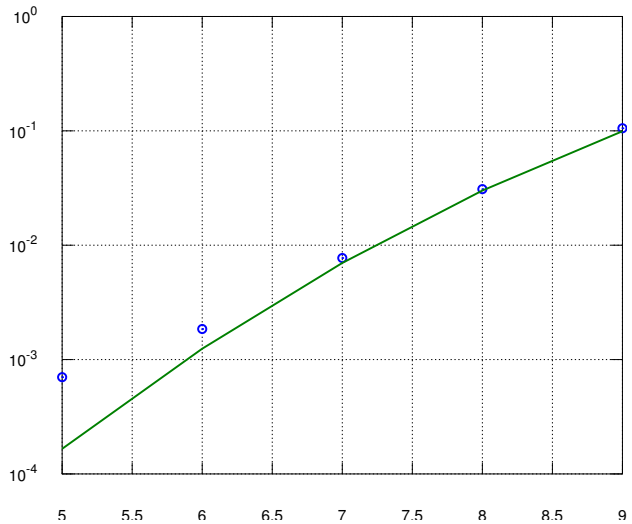


Figure 8. Comparison between the measured P_{eq} and the theoretical prediction for $P_\ell = 0.3$, $N_{\text{up}} = 13$ and $R \in \{6,7,8,9\}$.

Fig. 8 compares the measured equivalent packet loss probability P_{eq} (averaged over all the strata) in the case of fragment propagation with the probability of not receiving at least $R \in \{6,7,8,9\}$ packets out of $N_{\text{up}} = 13$ with a loss probability equal to $P_\ell = 0.3$. The match between theory and experiment is very good, the relatively large disagreement for $R = 6$ is due to the fact that the number of iterations (1000×20) is relatively small with respect to the expected value of P_{eq} ($\approx 10^{-3}$).

VII. Robustness against churn

An important problem in P2P streaming network is that a node can leave at any time, leaving its lower peers without data. The "turbulence" in the network induced by the random leaving of peers is called *churn*. Protecting a P2P streaming system from the effect of churn is a major goal in P2P system design. The effect of churn on PPETP was originally analyzed in [34]. In this section after recalling, for the sake of completeness, some results from [34], we simplify the results of [34] by giving some bounds that can make the design of a PPETP network easier.

Consider a network where each node is supposed to have N_{up} upper peers and let $H(t) \in \{0,\ldots,N_{\text{up}}\}$ denote the actual number of upper peers of a given node at time t. Note that $H(t)$

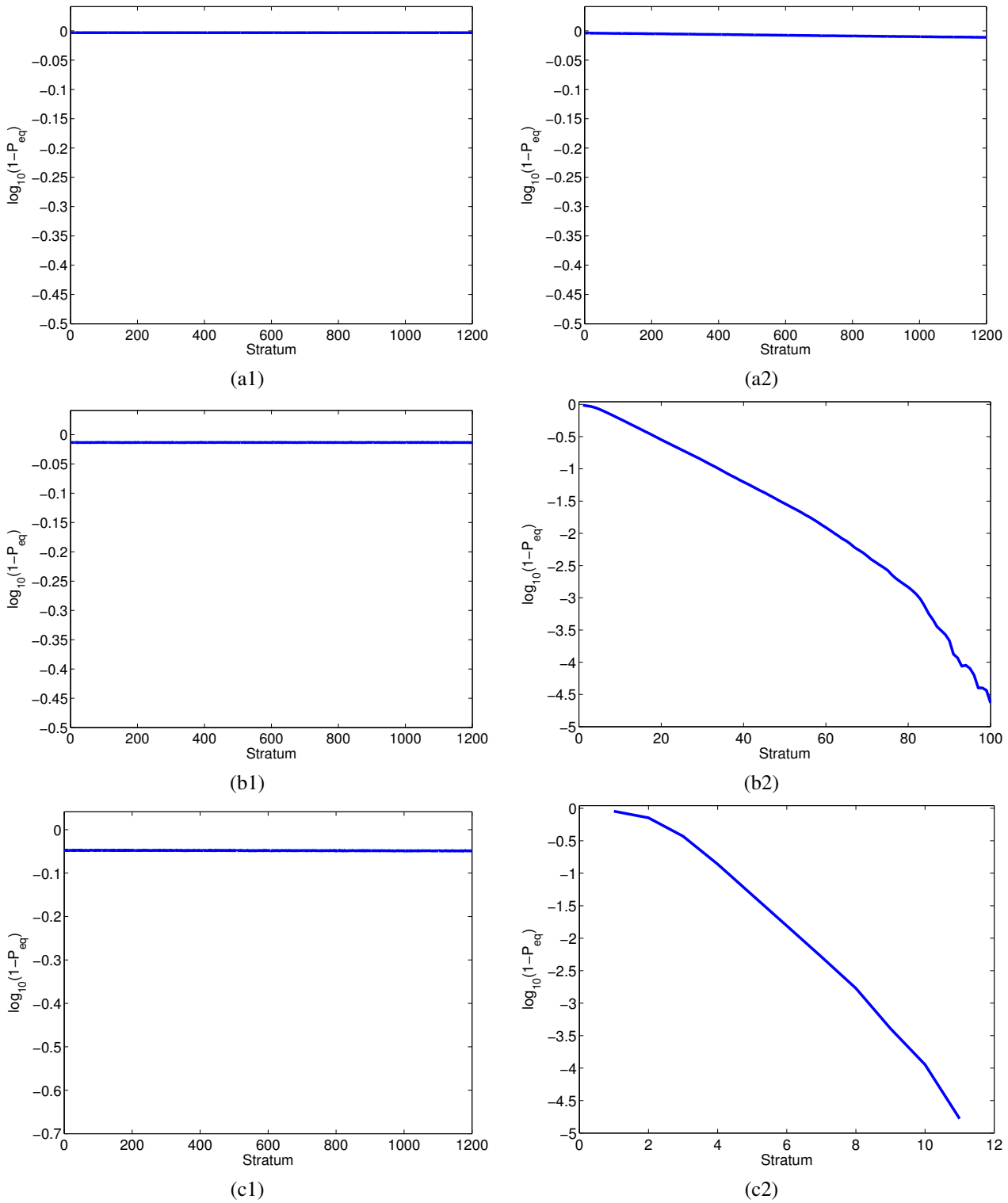


Figure 7. Probability of content packet recovery as function of the stratum number. All the plots are relative to $P_\ell = 0.2$ and $N_{\text{up}} = 13$. (a1) $R = 7$, with fragment propagation; (a2) $R = 7$, without fragment propagation; (a1) $R = 8$, with fragment propagation; (b2) $R = 8$, without fragment propagation; (a1) $R = 9$, with fragment propagation; (c2) $R = 9$, without fragment propagation. Plots (b2) and (c2) have a reduced range on the x axis since otherwise the plot would have looked as a vertical line.

can be smaller than N_{up}, for example, after some upper peer leaves. Note that if $H(t) < R$ the node cannot receive enough reduced packets to recover the content. We will call this event an *underflow* event and we will denote its probability as P_{under}.

In [34] probability P_{under} is computed as a function of R and N_{up} supposing that

1) The search of a new peer requires a time that can be described by an exponential random variable, with appropriate parameter λ. The average $1/\lambda$ is typically in the order of a few seconds or fraction of seconds.
2) The time a peer remains connected can be described by an appropriate distribution. with average $1/\mu$. Typical values of $1/\mu$ are in the order of at least several minutes. Note that we do not suppose the distribution exponential since some results in [7] show that distributions other than exponential model can be more appropriate.

According to [34], P_{under} can be written as

$$P_{\text{under}} = P[H(t) < R] = \frac{\sum_{n=0}^{R-1} \frac{\gamma^n}{n!}}{\sum_{n=0}^{N_{\text{up}}} \frac{\gamma^n}{n!}} \tag{26}$$

where $\gamma = \lambda/\mu$. Note that in a typical case one can expect γ to be quite large, at least of the order of few hundreds, even thousands.

The formula above, albeit exact, can be inconvenient to use for design purposes. In the following we are going to give an upper bound to probability (26) that holds for large values of γ and depends on γ, R and N_{up} in a more intuitive way. The upper bound we are going to present is not in [34] and it is published here for the first time. We will need the following lemma that allows us to upper bound the sums in (26) with a single term as soon as γ is large enough.

Lemma 1. *For every $M \in \mathbb{N}$ and $\gamma > 2(M-1)$ the following inequalities hold*

$$\frac{\gamma^M}{M!} < \sum_{n=0}^{M} \frac{\gamma^n}{n!} < 2\frac{\gamma^M}{M!} \tag{27}$$

Proof: The following equality is well-known

$$\sum_{n=0}^{M} \frac{\gamma^n}{n!} = e^{\gamma} \frac{\Gamma(M+1,\gamma)}{M!} \tag{28}$$

where $\Gamma(M+1,\gamma)$ is the incomplete gamma function. In [35] it is shown that for $a > 1$, $B > 1$ and $x > (a-1)B/(B-1)$ the following inequalities hold

$$x^{a-1}e^{-x} < \Gamma(a,x) < Bx^{a-1}e^{-x} \tag{29}$$

Using inequalities (29) in (28) with $a = M+1$, $B = 2$ and $x = \gamma > (a-1)B/(B-1) = 2M$, it follows

$$\frac{e^{\gamma}(\gamma^M e^{-\gamma})}{M!} < e^{\gamma}\frac{\Gamma(M+1,\gamma)}{M!} < \frac{e^{\gamma}(2\gamma^M e^{-\gamma})}{M!} \tag{30}$$

From (30) the thesis follows. ■

We will give the upper bound in the specific case of $N_{\text{up}} \leq 2R$. The more general case is not much more difficult, but it gives rise to more complex expressions that partially spoil the simplification introduced by the upper bound. Note that the hypothesis $N_{\text{up}} \leq 2R$ is quite reasonable since it seems very unlikely to have the necessity of a number of upper peers that is more than twice the minimum.

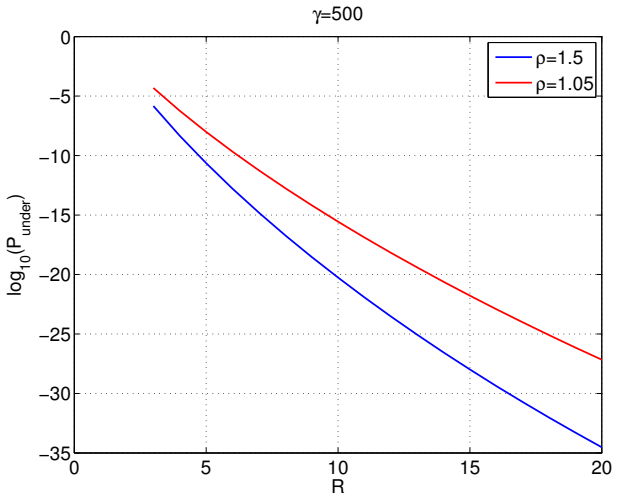


Figure 9. Upper bound to the underflow probability P_{under} for $\gamma = 500$, and $\rho = 1.05$ (the upper curve) and $\rho = 1.5$ (the lower curve).

Property 2. *If $N_{up} \leq 2R$ and $\gamma > 2(R-1)$, then the underflow probability P_{under} can be upper bounded as*

$$P_{under} \leq 2\left(\frac{N_{up}}{\gamma}\right)^{N_{up}-R+1} \tag{31}$$

Proof: From Lemma 1 with $M = R-1$, it follows that when $\gamma > 2(R-1)$,

$$P_{\text{under}} = \frac{\sum_{n=0}^{R-1}(\gamma^n/n!)}{\sum_{n=0}^{N_{\text{up}}}(\gamma^n/n!)} \leq \frac{\sum_{n=0}^{R-1}(\gamma^n/n!)}{\gamma^{N_{\text{up}}}/N_{\text{up}}!} \leq \frac{2\gamma^{R-1}/(R-1)!}{\gamma^{N_{\text{up}}}/N_{\text{up}}!} \tag{32}$$

By observing that

$$\begin{aligned} \frac{\gamma^{R-1}/(R-1)!}{\gamma^{N_{\text{up}}}/N_{\text{up}}!} &= \gamma^{R-N_{\text{up}}-1}R\cdots N_{\text{up}} \\ &\leq \gamma^{R-N_{\text{up}}-1}N_{\text{up}}^{N_{\text{up}}-R+1} = \left(\frac{N_{\text{up}}}{\gamma}\right)^{N_{\text{up}}-R+1} \end{aligned} \tag{33}$$

the thesis follows. ■

Note that since in a practical case γ will be of the order of many hundreds, while N_{up} is expected to be at most around ten, from Property 2 one can deduce that one can make P_{under} very small with a small number of excess peers $N_{\text{up}} - R$. Fig. 9 show two bounds for $\gamma = 500$, and $\rho = 1.05$ (the upper curve) and $\rho = 1.5$ (the lower curve).

VIII. Design Guidelines

The procedure for designing a PPETP network depends, of course, on the specific application and the corresponding figures of merit of interest. In this section we give some guidelines that can be useful in designing a PPETP network in

what can be expected to be a fairly common setup. Of course all the characteristics considered so far (e.g., robustness to packet loss, jitter, computational complexity, efficiency, ...) are interrelated one another and the trade-off between them will depend on the specific application.

We will suppose to have an estimate of the values P_ℓ (the loss probability over a single link), $1/\lambda$ (the average time a peer remains connected, see Section VII), $1/\mu$ (the average time required to find a new peer, see Section VII), the content bandwidth B and the minimum upload bandwidth $U_{\min}$ available at the nodes. We also have some quality of service constraints, such as a maximum underflow probability P_{under} (see Section VII) and a maximum packet loss probability at the application level P_{eq} (see Section VI). Finally, we desire to keep $\rho = N_{\text{up}}/R$ as small as possible, since the overall required bandwidth grows linearly with ρ. The parameters that we need to determine are the Galois field $\mathbb{F}_{2^d}$, the reduction factor R and the redundancy ρ (or, equivalently, the number of upper nodes $N_{\text{up}} = \rho R$).

An obvious constraint on R is given by the fact that, if data puncturing is not employed, it must be $R \geq \lceil B/U_{\min} \rceil$, where $\lceil x \rceil$ denotes the smallest integer not smaller than x.

Observe that the minimum value admissible for ρ is fixed by R and P_ℓ since ρ must be such that the probability of receiving at least R fragments out of ρR is not smaller than $1 - P_{\text{eq}}$. Note that for small values of P_{eq}, ρ cannot be smaller than $1/P_\ell$ and that it gets closer to that optimal value as R grows. Therefore, in order to minimize ρ, it is convenient to choose a large value for R.

Using a large R has other advantages, too. For example, both the probability P_{under} of the underflow event (see Section VII) and the the probability P_{fail} of having less than R different reduction parameters (see Section V-B) decrease with R. Moreover, for a fixed value of ρ, also the jitter (see Section V-F) and the decay of recovery probability $1 - P_{\text{eq}}$ (see Section VI) improve with R since $N_{\text{up}} = \rho R$. The only drawback of a large value of R is, according to (15), an increased computational complexity.

Summarizing, we can give the following guidelines for the choice of ρ and R

- Choose, tentatively, $d = 32$ since the increase in computational complexity is not huge (see Table I) and it helps in keeping P_{fail} small.
- Choose R as large as possible, at least large enough to satisfy the constraint $R \geq \lceil B/U_{\min} \rceil$.
- Choose ρ so the the probability of receiving at least R fragments out of ρR is not smaller than $1 - P_{\text{eq}}$.
- Verify that, with the given choices of ρ and R, the constraints on P_{fail} and P_{under} are satisfied. If they are not, choose a larger R. Note that even if we increase R, we can keep the same ρ since it will satisfy the P_{eq} constraint even if R is increased. Alternatively, if R cannot be increased because of the computational complexity, one can increase ρ.
- Verify that the computational complexity for the chosen d and R is acceptable. If it is not, one can lower it by using a different value for d (e.g., $d = 16$) or reiterating the design procedure with a smaller R. If R was already equal to the minimum value $\lceil B/U_{\min} \rceil$, one can try to employ data puncturing for the nodes with smallest bandwidth.

```
# Input  : cl and dl
# Output : al and ah

movb %cl, %dh
shll #1, %edx
movw tbl_4(%edx), %ax
```

(a)

```
# Input  : cl and dl
# Output : al

movb %cl, %dh
movb tbl_8(%edx), %al
```

(b)

```
# Input  : cx, dx
# Output : ax

# compute log(%ecx)
movl %ecx, %eax
orw  %ax, %ax
jz   done
shl  #1, %eax
movw log_16(%eax), %cx

# compute log(%edx)
movl %edx, %eax
orw  %ax, %ax
jz   done
shl  #1, %eax
movw log_16(%eax), %dx

# Compute the sum of
# the logs mod 2**16-1
addw %dx, %cx
jnc no_mod_needed
incw %cx

no_mod_needed:
   shll #1, %ecx
   movw exp_16(%ecx), %ax
done:
```

(c)

```
# Input  : ecx, edx
# Output : eax

carry_mask=0x8299
# Init the result
movl #0, %eax

orl  %edx, %edx
jz   done

main_loop:
   # If the LSB of
   # edx is 1, xor
   # ecx with the result
   test #0x0001, %edx
   jz  skip_xor
   xorl %ecx, %eax

skip_xor:
   # Shift left cx
   shl #1, %ecx
   jnc no_reduction
   xor carry_mask, %ecx

no_reduction:
   # Shift right dx
   shr #1, %edx
   jnz  main_loop
done:
```

(d)

Figure 10. The product algorithms used in the velocity tests. (a) Algorithm for $\mathbb{F}_{2^4}$. (b) Algorithm for $\mathbb{F}_{2^8}$. (c) Algorithm for $\mathbb{F}_{2^{16}}$. (d) Algorithm for $\mathbb{F}_{2^{32}}$.

IX. Conclusions and Future Work

This article has described PPETP, an overlay multicast protocol that allows for efficient data propagation even when some nodes have limited resources. A quantitative analysis of some figures of merit of PPETP and some design guidelines have been presented.

A. Acknowledgments

PPETP is partially funded by Italian Ministry PRIN project *Arachne*.

Appendix

A. Computation of P_{fail}

Our goal is to compute $P_{\text{fail}}(N_{\text{up}}, R, S)$, that is the probability that after N_{up} drawing from an alphabet with S elements we have less than R different values. This experiment can be represented by the finite state system shown in Fig. 12. Each state is labeled with the number of different symbols extracted

```
uint16_t table[65536];  /* Filled at init time */

void F16_mult(byte b,
              byte in_1,   byte in_2,
              byte *out_1, byte *out_2)
 {
   /* Make the 16-bit index to access the table
    * as follows
    *
    *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    *     |  r_2  |  r_1  |       b       |
    *     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    *    MSB                             LSB
    */
   uint16 index    = in_2 << 12 + in_1 << 8 + b;
   uint16 out_pair = table[index];

   *v_1 = out_pair & 0xff;  /* LS bits */
   *v_2 = out_pair >> 8;    /* MS bits */
 }
```

Figure 11. Pseudo-C code to compute with a single table access products `out_1 = in_1*b` and `out_2 = in_2*b`, where `in_1` and `in_2` represent elements of $\mathbb{F}_{16}$ and `b`, `out_1` and `out_2` vectors in $\mathbb{F}_{16}^2$.

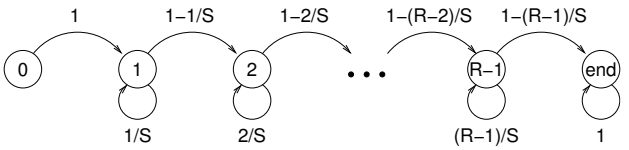


Figure 12. Markov chain used to compute the curves of Fig. 4.

so far and the system starts from state 0. The system goes in the state marked with "end" when R or more different symbols have been extracted. It is easy to see that $P_{\text{fail}}(N_{\text{up}},R,S) = 1-P[s(N_{\text{up}}) = \text{end}]$, where $s(n)$ is the state after n extractions. This probability can be easily obtained from $\mathbf{P}^{N_{\text{up}}}$, where $\mathbf{P}$ is the transition matrix of Fig. 12. By writing explicitly $\mathbf{P}$ it is easy to check that the $\lim_{n\to\infty} P[s(n) = \text{end}] = 1$ and that the largest eigenvalue of $\mathbf{P}$ less than 1 is $(R-1)/S$. Therefore, $P_{\text{fail}}(n,R,S)$ converges to zero as $[(R-1)/S]^n$.

B. Proof of Property 1

In order to proof Property 1 we need the following lemma.

Lemma 2. *For every $K \geq 1$, the probability that stratum K is in full state is bounded as follows*

$$P[\mathbf{F}_K = \Omega] \geq \lambda^{\Sigma_{n=1}^{K} L_n} \tag{34}$$

where λ is as in (21).

Proof: We proceed by induction. For $K = 1$ the event $\mathbf{F}_1 = \Omega$ holds if every node of the first stratum receives at least T fragments. Since the upper peers of the nodes of the first stratum are origin servers, the number of fragments received by a node is a binomial variable with N_{up} tentatives and success probability P_T, that is, the probability that a given node of the first stratum is in firing state is λ. Since all the links from stratum 0 to stratum 1 are independent one another,

$$P[\mathbf{F}_1 = \Omega] = \lambda^{L_1} \tag{35}$$

that is (34) with the equality sign.

Suppose now that bound (34) holds for $K-1$ and prove it for $K > 1$. It is

$$\begin{aligned} P[\mathbf{F}_K = \Omega] &= \sum_{\mathbf{u}\in\{0,1\}^{L_{K-1}}} P[\mathbf{F}_K = \Omega|\mathbf{F}_{K-1} = \mathbf{u}]P[\mathbf{F}_{K-1} = \mathbf{u}] \\ &\geq P[\mathbf{F}_K = \Omega|\mathbf{F}_{K-1} = \Omega]P[\mathbf{F}_{K-1} = \Omega] \\ &\geq P[\mathbf{F}_K = \Omega|\mathbf{F}_{K-1} = \Omega]\lambda^{\Sigma_{n=1}^{K-1} L_n} \end{aligned} \tag{36}$$

where the last inequality follows from the inductive hypothesis.

In order to compute $P[\mathbf{F}_K = \Omega|\mathbf{F}_{K-1} = \Omega]$ observe that if all the nodes in stratum $K-1$ are in firing state, a reasoning similar to the one used to derive (35) holds and one obtains

$$P[\mathbf{F}_K = \Omega|\mathbf{F}_{K-1} = \Omega] = \lambda^{L_K} \tag{37}$$

Using (37) in (36) gives the thesis. ■

References

[1] R. Bernardini, R. C. Fabbro, and R. Rinaldo, "Ppetp: A peer-to-peer overlay multicast protocol for multimedia streaming," in *Proc. of CONTENT 2011*, (Rome), Sept. 2011. Best paper award.

[2] E. Adar and B. A. Huberman, "Free riding on Gnutella," *First Monday*, vol. 5, October 2000.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, (New York, NY, USA), pp. 407–418, ACM, 2003.

[4] V. Fodor and G. Dán, "Resilience in live peer-to-peer streaming," *IEEE Communications Magazine*, vol. 45, pp. 116–123, June 2007.

[5] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proc. of NOSSDAV 2002*, (Miami, Florida, USA), ACM, May 2002.

[6] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?," in *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, (Cambridge, MA), USENIX, April 2007.

[7] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, (Rio de Janeriro, Brazil), pp. 189–202, SIGCOMM, 2006.

[8] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, pp. 1655–1666, December 2007.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *IN PROC. ACM SIGCOMM 2001*, pp. 161–172, 2001.

[10] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *19th ACM Symposium on Operating Systems Principles, 2003*, 2003.

[11] S. Marti and H. Garcia-molina, "Taxonomy of trust: Categorizing p2p reputation systems," *Computer Networks*, vol. 50, pp. 472–484, 2006.

[12] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *12th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pp. 1–10, July 2002.

[13] Y. Yue, C. Lin, and Z. Tan, "Analyzing the performance and fairness of bittorrent-like networks using a general fluid model," *Computer Communications*, vol. 29, no. 18, pp. 3946 – 3956, 2006.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, August 2001.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, Nov. 2001.

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

[17] D. A. Tran, K. A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA), 2003.

[18] V. Padmanabhan, H. Wang, and P. Chou, "Supporting heterogeneity and congestion control in peer-to-peer multicast streaming," in *Proceedings of IPTPS*, (San Diego, CA), Feb. 2004.

[19] X. Liao, H. Jin, Y. Liu, L. Ni, and D. Deng, "Chunkyspread: Heterogeneous unstructured tree-based peer to peer multicast," in *Proceedings of IEEE International Conference on Computer Communications*, IEEE Computer Society, Apr. 2006.

[20] 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.

[21] L. X., J. H., L. Y., N. L., and D. D., "AnySee: Peer-to-Peer live streaming," in *Proc. INFOCOM 2006*, pp. 1–10, 2006.

[22] N. Magharei and R. Rejaie, "Prime: peer-to-peer receiver-driven mesh-based streaming," in *Proceedings of IEEE International Conference on Computer Communications*, (Alaska), May 2007.

[23] J. Liang and K. Nahrstedt, "Dagstream: locality aware and failure resilient peer-to-peer streaming," in *Proceedings of MMCN*, Jan. 2006.

[24] C. Wu and B. Li, "rStream: resilient and optimal peer-to-peer streaming with rateless codes," *T-par*, pp. 77–92, Jan. 2008.

[25] A. Magnetto, R. Gaeta, M. Grangetto, and M. Sereno, "P2p streaming with lt codes: a prototype experimentation," in *Proc. ACM Multimedia 2010*, pp. 7–12, Oct. 2010.

[26] R. Bernardini, R. C. Fabbro, and R. Rinaldo, "Peer-to-peer epi-transport protocol." `http://tools.ietf.org/html/draft-bernardini-ppetp`, Jan. 2011. Internet Draft, work in progress.

[27] R. Bernardini, R. C. Fabbro, and R. Rinaldo, "Group based reduction schemes for streaming applications," *ISRN Communications and Networking*, vol. 2011, 2011. Article ID 898254, doi:10.5402/2011/898254.

[28] R. Bernardini, R. C. Fabbro, and R. Rinaldo, "Peer-to-peer streaming based on network coding improves packet jitter," in *Proc. of ACM Multimedia 2010*, (Florence, Italy), Oct. 2010.

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

[30] http://www.gnu.org/software/binutils.

[31] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*. No. 253669-033US, December 2009.

[32] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, November 1979.

[33] T. M. Cover and J. A. Thomas, *Information theory*. New York: Wiley, 1991.

[34] 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.

[35] P. Natalini and B. Palumbo, "Inequalities for the incomplete gamma function," *Math. Inequal. Appl. 3*, no. 1, pp. 69–77, 2000.