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 a protocol especially designed for peer-to-peer data distribution to a large number of users. The protocol is suited for the efficient distribution of live multimedia and it can exploit even the limited resources of residential users. Special care was paid to make the protocol *back-compatible* with existent multimedia tools and protocols, so that software and protocols already multicast-enabled require only minor changes to be adapted to the new protocol. The flexibility, the openess and the features of the proposed protocol makes it an interesting solution for streaming content to large audiences.**

***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 $6 \cdot 10^8$).

The upload bandwidth problem is not limited to the "large audience" scenario, but it can also be found at smaller scales. Consider, for example, the case of a medium-size community with 100–1000 members (e.g., a political party or a fan club) that wants its own IPTV channel to stream events to its members or, maybe, organize virtual meetings. If the association aims to "YouTube quality" video (hundreds of Kbit/s), the overall bandwidth is in the order of hundreds of Mbit/s. Although this is within current technology capabilities, the implementation of such services could prove too expensive for the association.

Multicast is of course a possible solution, but it has its drawbacks too. Maybe the most difficult issue in using multicast, in applications like these, is that the audience is expected to be spread among several different Internet Service Providers (ISPs) and multicast across different Autonomous System (AS) is not trivial, both on a technical and on a administrative side.

An approach that recently attracted interest in the research community is the use of peer-to-peer (P2P) solutions [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14]. 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. This could be beneficial to both the "large audience" and the "fan club" scenarios.

Unfortunately, the application of the P2P paradigm to multimedia streaming has some difficulties such as

- **Asymmetric bandwidth** Depending on the media type and quality, the typical residential users, connected to the Internet via an ADSL, could have enough download bandwidth to receive the stream, but not enough upload bandwidth to retransmit it. Therefore, the solution of having the user retransmit the content to another user is not applicable and more sophisticated solutions are needed.
- **Heterogeneous nodes** The network can include nodes with different upload capabilities, from residential users with few hundreds kbit/s up to nodes with upload bandwidth of several Mbit/s. A good P2P structure should be able to exploit the bandwidth of each peer as much as possible, both for low-bandwidth and high-bandwidth nodes.
- **Sudden departures** A node can leave the network at any time, possibly leaving other nodes without data for a long time.
- **Security** P2P networks have several security issues [15]. Here we simply cite the *stream poisoning attack* where a node sends incorrect packets which cause an incorrect decoding and are propagated to the whole network by the P2P1 mechanism.
- **Network Address Translators (NAT)** Several residential users are behind at least the NAT built-in in their modem and this is a problem for P2P solutions since the NAT makes the user PC unreachable by outside peers.

This article describes the *Peer-to-Peer Epi-Transport Protocol* (PPETP), a peer-to-peer protocol developed as part of the project *Corallo* hosted on *SourceForge*. This paper is organized as follows: Section II describes the design goals that guided the development of PPETP, Section III gives an overview of PPETP and introduces some "PPETP jargon," Section IV describes the idea of "reduction procedure" that is at the core of PPETP, Section V shows the similarities between PPETP and IP multicast and how these similarities allow one to reuse with PPETP all the tools (protocols, formats, and so

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-157-1

on) developed for multicast with a minimal change, Section VI illustrates some practical examples of use, Section VII shows some results about some performance aspects of PPETP, Section VIII presents the conclusions.

II. Design goals

Our objective was to design a protocol that would solve the problems enumerated above, that could be used in several applicative contexts (not just video streaming) and that would look, from the application level, like another transport protocol. More precisely, our design goals were

- **Multicast-like structure** From the application level the protocol must look like a multicast protocol, with an API (Application Programming Interface) similar to the well-known BSD sockets. This would simplify the integration of the new protocol in existing protocols and applications.
- **Usability with heterogeneous networks** The system must be able to exploit efficiently the bandwidth of each user, both for high- and low-upload bandwidth users.
- **Robustness with respect to data losses** In particular, the video must not stop even if one or more peer suddenly leave the network.
- **Security** The protocol must counteract possible attacks that a malicious user could try. In particular, it must be difficult to poison the data stream and, in case of an attack, it should be possible to find out the culprit.
- **Usability with NAT** The protocol must take care by itself of the possible presence of NATs. The application programmer should not worry about the presence of NATs.
- **Usability with any data type** Like a true transport protocol, the developed protocol must be able to carry any type of data.
- **Flexible topology** The protocol must not be tied to a specific network topology, nor to a specific peer-discovery technique, but it must be possible to use it with different peer discovery procedures and topologies.

Remark II.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.

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 [16].

A PPETP network is made of several nodes that exchange data and control information over a non necessarily reliable protocol (currently PPETP is built on UDP, but other protocols, such as the Data Congestion Control Protocol [17], can be added in the future). Since each node streams autonomously to a (fairly stable) set of 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. (This nomenclature is inspired to the typical picture of a tree structured network with data flowing from top to bottom).

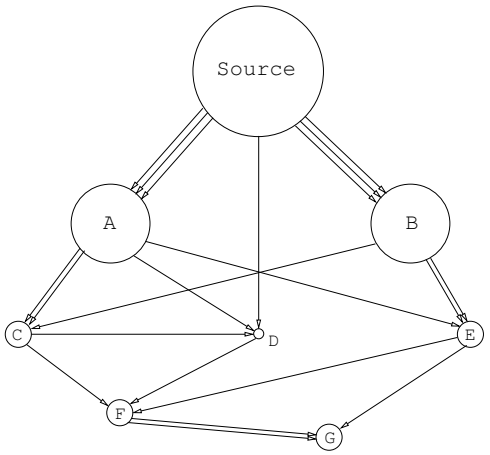

Figure 1. Example of a PPETP network for multimedia streaming.

A key characteristic of PPETP is that every node does not upload to other nodes the whole content stream, but a reduced version of it that requires less bandwidth. The details of how the reduced streams are produced are described in the following. Here it suffices to say that the original content can be recovered as soon as the node receives a minimum number $N_{\min}$ (typically chosen off-line) of reduced data. It follows that in a typical PPETP network a node has many upper peers and, possibly, many lower peers.

Fig. 1 shows an example of a possible PPETP network for multimedia streaming with $N_{\min} = 3$. Each arrow represents a reduced stream, each circle represents a node and the node 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 reduced streams. Note also that the source "feeds" directly nodes A and B by sending them three different reduced streams. Finally, note that the network of Fig. 1 is an irregular mesh, showing that not only tree-structured networks are possible with PPETP. As already said, by design PPETP does not mandate any particular network topology nor any specific way to find the peers. The specific application can construct the network as it sees fit.

IV. Data reduction procedures

As said above, the cornerstone of PPETP is a special type of network coding called *reduction procedure*. The idea is that, in order to take an advantage of even small upload bandwidths, a node does not propagate the multimedia packets, but a *reduced version* of them obtained by processing each packet with a *reduction function*. The result of the reduction function is a *reduced packet* whose size is (typically) a fraction of the size of the original packet. Thus, the reduced packets

transmission can fit the limited upload bandwidth of each peer. The *reduction factor*, that is the ratio between the size of the content packet and the size of its reduced version, will be denoted with R and supposed approximately constant (we say *approximately* since we admit slightly variations due, for example, to padding).

The reduction function is parametrized by a *reduction parameter* so that different reduced versions of the packet can be obtained by processing the same packet using different reduction parameters. The reduction procedure has the property that a node can reconstruct the original content packet as soon as it knows a sufficient number of reduced versions of the packet itself.

An overview of the typical behavior of a PPETP node is the following: the node, after receiving at least $N_{\min}$ reduced versions of the same content packet, recovers the packet itself and moves it toward the application level. Moreover, if the node has some lower peers, it reduces the recovered packet and sends the reduced versions to its lower peers. Nodes with larger upload bandwidth can serve several peers and also send to the same peer several different reduced streams, as exemplified in Fig. 1.

Example IV.1 (An example of reduction procedure)
The description of reduction function given above is very general and abstract. This follows the specification of PPETP that, for the sake of future extensions, does not impose a specific reduction procedure, but demands its description to documents called *reduction profiles*.

Since the abstract idea of a reduction procedure could be difficult to grasp on an intuitive level, we cite as an example of reduction procedure the algorithm described in [18] (used in the *Vandermonde* reduction profile).

To reduce the size of a content packet by a factor R, the packet is interpreted as a vector with entries b_0, b_1, ... in $\mathrm{GF}(2^d)$ (the finite field with 2^d elements) and every R-tuple of values b_0, b_1, ..., $b_{R-1} \in \mathrm{GF}(2^d)$ is replaced by

$$c_r = b_0 + rb_1 + \cdots + r^{R-1} b_{R-1}$$

where r is an element of $\mathrm{GF}(2^d)$ randomly chosen by (or assigned to) the node at start-up. The reduced packet is obtained by concatenating the values c_r obtained as above. Note that since this procedure replaces a sequence of R elements with a single element, the required upload bandwidth will be R times smaller than the bandwidth of the multimedia content.

In order to recover the original content packet a node contacts at least R peers and receives from them their reduced packets. It is easy to show that if each peer chooses a different value for r, then the node can recover the original values b_0, b_1, ..., by solving a linear system associated to a Vandermonde matrix [18].

A. Reduction profiles

The reduction procedure described above is not the only possible approach for data reduction. For example, other network coding procedures (e.g., digital fountains) could be used. In order to allow for future adoptions of different techniques, PPETP does not define a specific reduction procedure, but demands such a definition to side documents called *reduction profiles*. This makes possible to extend PPETP with new reduction procedures without changing its core definition.

Two profiles currently are defined: the Vandermonde profile (described above) and the Basic profile that does no reduction at all and it is thought for streams with very low bandwidth (e.g., RTCP streams) where the bandwidth saving would not be worth the additional complexity of a "true" reduction profile.

Although PPETP does not mandate any special characteristic to a reduction profile, it is expected that future reduction profiles will share with the Vandermonde profile the following important characteristics.

- **Size reduction** The size of the reduced packet is a fraction ($\approx 1/R$) of the size of the original content packet.
- **Parametrization** The reduction procedure depends on a set of parameters. Using different parameters gives rise to different reduced versions of the content packet. In the Vandermonde profile the reduction parameter is the value r.
- **Reconstruction** The content packet can be recovered from the knowledge of a suitable number $N_{\min}$ of different reduced versions (intuitively, $N_{\min} \geq R$). In some cases, such as in the Vandermonde profile describe above, $N_{\min} = R$, but this can be different in other profiles. For example, in an hypothetical reduction profile based on digital fountains, $N_{\min}$ would be a random variable with average slightly larger than R. In the following, for the sake of simplicity, we will suppose $N_{\min}$ deterministic.

B. Consequences of the reduction procedure

The reduction procedure in PPETP allows us to meet some of the previously stated design goals.

a) Exploitation of low-bandwidth nodes: Since the size of a reduced packet is a fraction of the size of the original content packet, the corresponding upload bandwidth is a fraction of the bandwidth of the content stream.

For very small upload bandwidths (that would required too large reduction factors) PPETP allows to introduce a *puncturing* that can be *random* (the packet is sent with a given probability) or *deterministic* (packets are sent according to a pattern). A careful use of puncturing allows for a finer control of the upload bandwidth.

b) Usage with heterogeneous networks: It is easy to manage the case when nodes have different bandwidths. For example, nodes with large upload bandwidth can serve several peers, Moreover, nodes with a large upload bandwidth can produce different reduced streams (by using different reduction parameters with the same content packet) and send more than one stream to the same lower peer (see Fig. 1).

c) Robustness to data loss: To counteract the risk of packet losses (due, e.g., to network congestion or peer leaving) the node requests data to $N > N_{\min}$ peers and recovers the content as soon as it receives $N_{\min}$ packets.

d) Security: To prevent stream poisoning, the node requests data from $N > N_{\min}$ peers, recovers the packet using $N_{\min}$ reduced packets and checks that the remaining packets are coherent with the reconstructed packet. This procedure can counteract a coordinated attack from $N - N_{\min}$ peers and, with a slightly variation, it allows to find (and punish) the node(s) that tried the attack.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-157-1

If we punish who tries a poisoning attack, a malicious user could try a *defamatory attack* by sending corrupt packets while pretending to be another peer. In order to avoid this type of attack, PPETP allows a node to sign the packets that transmits.

e) Distributed parameter assignment: The reduction parameter used by the node can be assigned by an external entity or it can be chosen autonomously by the node (maybe at random). The latter is especially interesting, since it does not require a centralized actor for parameter assignment. The probability that two peers choose the same parameter can be made negligible by using a large enough parameter space.

f) Independence on the data type: Since the reduction procedure handles the content packets just as "sequences of bits," PPETP can be used with any type of data (e.g., audio/video, encoded with scalable or multiple description encoders, even encrypted data).

V. PPETP AND MULTICAST

As said above, one of our objectives was to design a protocol that looked, from the application point of view, like a multicast protocol. A first step toward this objective was the development of a protocol such that all the "P2P-related matters" (e.g., data reduction and reconstruction, handshaking with new peers and so on) could be handled inside the library implementing PPETP. In this way, the PPETP API could be made similar to the BSD socket API.

Moreover, in order to make the integration of PPETP with existing protocols (e.g., SDP [19], RTSP [20], SIP [21]) simpler, we decided to introduce the concept of *address* of a PPETP session in the form of an (host, port) pair. The problem is that a PPETP network, being distributed, has not a "natural" address. However, since a PPETP session needs to be configured (e.g., to set the reduction profile, the reduction parameters or any cryptographic credential used to communicate with other peers), the host part was chosen to be the address of a *configuration server* used to get the configuration data. (Note that every P2P network needs at least a "starting point" used by users to join the network; the starting point for a PPETP network is the configuration server.) The role of the port is played by the *session number*, a 16-bit integer that, together with the host address, uniquely identifies the PPETP session.

The configuration server is queried via a special protocol designed to be light-weight and stateless, so that it is less prone to Denial-of-Service (DoS) attacks and it can handle also a large number of connections. If needed, the configuration server can redirect the users, after authentication, to a more powerful protocol (e.g., an HTTP-based one) or maybe a distributed one (where the configuration data are obtained from others peers).

VI. EXAMPLES OF USE

In order to make clearer the just given overview of PPETP, this section describes two possible typical uses of PPETP: a live streaming application and a conference application.

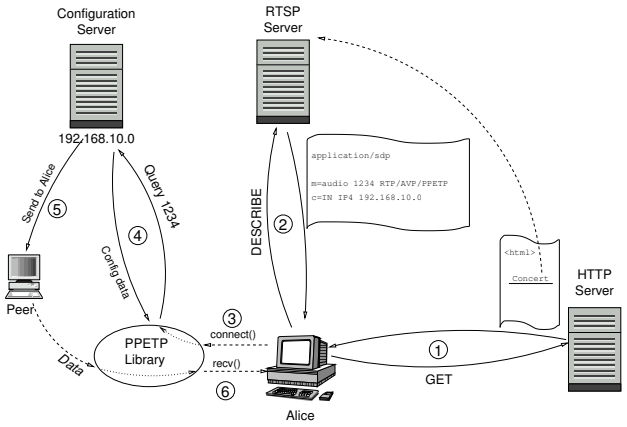

Figure 2. Example of establishment of a PPETP session

A. Live streaming

Suppose Alice wants to watch a concert streamed over PPETP. A possible sequence of actions is the following (see also Fig. 2)

1) Alice goes to the web page of the streamer, finds a link related to the concert and clicks on it.
2) The link points to an RTSP server. The browser launches a "viewer" that queries the RTSP server to get a program description (in SDP format [19]) that says that the program is streamed over PPETP.
3) The viewer opens a PPETP socket and connects it to the session address found in the SDP description by using a function similar to the BSD `connect()`.
4) The connection function queries the configuration server that replies with the configuration data.
5) Now Alice's upper peers must be notified to send data to Alice. This can be done in several ways, for example
   a) The PPETP network is fully managed by the video provider. In this case, the configuration server chooses the upper peers and asks them (via suitable control packets) to send data to Alice. If an upper peer is behind a NAT, the control packet will also cause the initiation of a suitable NAT traversal procedure. This is the case shown in Fig. 2.
   Although this centralized solution could seem to introduce a "single point of failure," it must be said that in this case there is an actor (the streaming provider) that is interested in doing the streaming. If the provider's host fails, the whole system makes no sense. Moreover, this centralized solution allows for a finer control of some PPETP network characteristics such as the quality of service assigned to Alice and the locality of the network.
   b) The server chooses the upper peers, sends the list to Alice and let her contact the peers.
   c) The server sends to Alice a list with some possible peers. Alice contacts few peers, asking for data; if a peer has no more bandwidth available, it refuses the request and Alice tries another peer until she gets enough peers. Note that, with this setup, it is difficult to make sure that Alice gets only its fair

share of resources.

d) A possible "strongly distributed" solution is the following: the nodes are organized in a Distributed Hash Table (DHT) where a set of "keys" (e.g., b-bit integers) is assigned to each node. The address of the DHT "entry points" is included in the configuration data. Alice randomly draws few keys, searches for the corresponding nodes and contacts them. Nodes that run out of bandwidth, refuse the request.

6) Alice receives reduced data. As soon as enough data are available, content packets are recovered and moved to the application level. The viewer reads the recovered data by means of a function similar to the BSD `recv()`, gives the data to the decoder and the result is shown to the user.
7) Suppose now that Bob joins the network and that the server assigns him Alice as an upper peer. Alice's host will receive a control packet that asks to send data to Bob.
8) In response to the received request, Alice's host applies the reduction procedure to the recovered packets and sends the result to Bob.
9) When Alice wants to stop to watch the concert, sends a TEARDOWN request to the RTSP server that in turn sends suitable control packets to Alice's upper peers, asking them to stop the transmission toward Alice and maybe redirecting them to the lower peers of Alice. Alternatively, Alice herself can redirect her upper peers to her lower peers.
10) If Alice suddenly leaves (maybe because of a bug), her lower peers notice her absence because they stop receiving data from her. As a consequence of this, they search for new peers. Note that if the network was built with redundancy, the *users* associated to Alice's lower peers *would not notice* the sudden departure since they will keep receiving enough data to recover the content packets.

B. Conferencing

The multicast-like nature of PPETP makes it an interesting solution for conferences. Conference management can be done via SIP [21], including in the session description the address of the PPETP session. In this case, every node "injects" its data on the network via a function similar to the BSD `send()` and reads from the PPETP socket the packets produced by the other nodes. The problem of separating the packets according to their source is outside the scope of PPETP and it pertains to the application. For example, if RTP is used, packets can be partitioned according to their SSRC [22].

C. Comments to the examples

It is worth to emphasize that most of the P2P management (e.g., NAT traversal, handshaking with the new peer) is handled by the PPETP library and it does not arrive at the application level. It should be clear from the examples above that the application just needs to (i) open a PPETP socket and connect it to the PPETP address, (ii) read/write data from/to it and (iii) close it when done.

Note also that since for PPETP a packet is just "a collection of bytes," any type of data can be transmitted over PPETP. This means that all the currently available streaming tools (e.g., RTP, RTCP, audio/video coders, scalable or multiple description coding, encryption procedures) can be transparently used with PPETP.

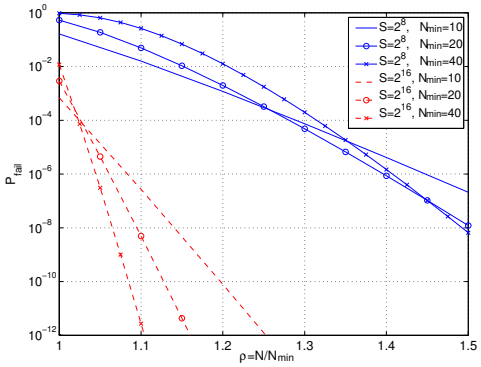

Figure 3. Failure probability P_{failure} as function of the redundancy ρ and the parameter space size S.

VII. PPETP Performance

In this section we give some figures about the performance of PPETP. For the sake of simplicity we will suppose that $N_{\min} = R$ (as it happens with the *Vandermonde* profile).

g) Failure probability: If each peer chooses at random its reduction parameters, it could happen that the set of reduction parameters associated to the N upper peers of a node has less than $N_{\min}$ different parameters (*failure event*) and the node will never be able to recover the content packets. Intuitively, the probability P_{failure} of such event gets lower when the redundancy ratio $\rho = N/N_{\min}$ or the parameter space size get larger. This intuition is confirmed by Fig. 3 that shows P_{failure} as a function of the redundancy ρ and of the parameter space size S.

h) Robustness to sudden departures: In order to give a feeling of the robustness to departures offered by PPETP, assume that a node has $N > N_{\min}$ upper peers and as soon as a peer leaves the node searches for a new one. If too many peers leave in a short time, the node could remain with less than $N_{\min}$ upper peers (*underflow event*).

The probability P_{under} of the underflow event can be computed by modeling the set of upper peers as a queue with N servers, system size N, mean inter-arrival time (i.e., the time required to find a new peer) equal to T_{find}, and mean service time (i.e., the mean time a node remains connected) equal to T_{leave}. With this model, P_{under} is the probability of having less than $N_{\min}$ peers in the system.

A plot of P_{under} as function of ρ and $N_{\min}$ when $T_{\text{leave}}/T_{\text{find}} = 70$ and with the hypothesis of Poisson arrival times can be seen in Fig. 4 [18]. Probability P_{under} gets smaller when ratio $T_{\text{leave}}/T_{\text{find}}$ gets larger.

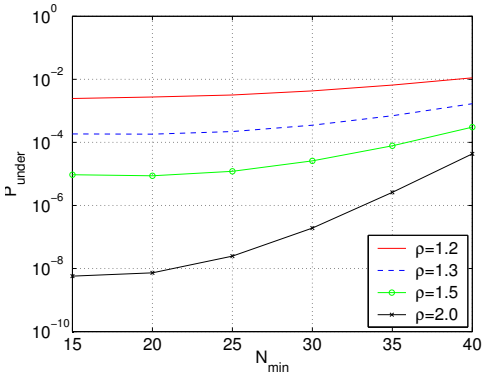

Figure 4. Underflow probability P_{under} as function of $N_{\min}$ for $T_{\text{leave}}/T_{\text{find}} = 70$.

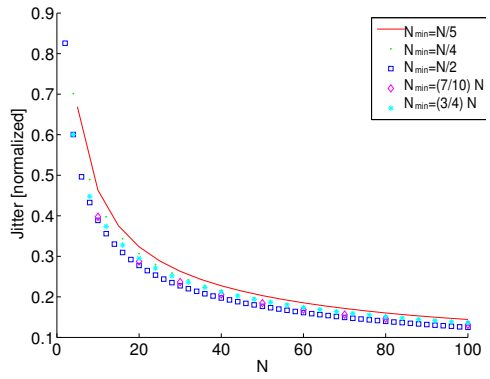

Figure 5. Jitter as a function of N (peer delays distributed as $\mathcal{N}(0,1)$)

i) Jitter reduction: A nice side effect of the use of network coding in PPETP is the reduction of the jitter observed by the node. Intuitively, this happens because the time a content packet is recovered is the time necessary for the arrival of the $N_{\min}$ fastest packets out of N. Fig. 5 shows the theoretical prediction of the jitter (i.e., the standard deviation of the reconstruction time), as a function of $N_{\min}$ and N, when the delays are Gaussian with mean m and variance σ^2. The values on the vertical axis are measured in units of σ. Note that the jitter decays as $1/\sqrt{N}$ [23]. This behavior was also verified experimentally [24].

VIII. Conclusions

This article has described PPETP, an overlay multicast protocol that allows for efficient data propagation even when some nodes have limited resources. The protocol is designed to appear at the application level as a multicast protocol, allowing for its easy inclusion in existing protocols and software. PPETP is robust against packet losses and it has tools that help counteracting possible attacks such as stream poisoning or DoS tentatives. PPETP is currently hosted by *SourceForge* as part of the open source project *Corallo*.

A. Acknowledgments

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

References

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

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)." RFC 4340 (Proposed Standard), Mar. 2006.

[18] 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.

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

[20] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)." RFC 2326 (Proposed Standard), Apr. 1998.

[21] 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.

[22] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." RFC 3550 (Standard), July 2003.

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

[24] 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.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-157-1