ACCEPTED FROM OPEN CALL

Using Digital Fountains in Future IPTV Streaming Platforms: A Future Perspective

Alessandro Cattaneo, Alexandro Sentinelli, Andrea Vitali, and Luca Celetto, STMicroelectronics
Gustavo Marfia and Marco Roccetti, University of Bologna
Mario Gerla, University of California

ABSTRACT

Coding techniques are progressively succeeding in finding a beneficial use throughout all the components of advanced multimedia streaming networks, including peer-to-peer (P2P) content distribution strategies. In fact, although P2P algorithms are regarded as an important technology that will be part of future commercial Internet protocol television (IPTV) platforms, it has been shown that their ability to efficiently distribute multimedia streams within strict time constraints, and their robustness to high churn rates, can still improve when putting to good use source and network coding algorithms. For this reason, in the past few years a wealth of research has been undertaken aiming at jointly optimizing both coding and streaming schemes. In this article, we analyze the progress made by the latest scientific research in the field of jointly using coding and P2P streaming strategies, and we offer a set of experimental results performed on a real planet-wide P2P streaming test-bed, using Digital Fountains (DFs). Surprisingly, our test-bed study reveals that utilizing DF techniques is not always beneficial to the distribution of streaming data using a P2P distribution strategy. In fact, our results show that only when the network scales beyond a certain size, it is possible to appreciate the advantages introduced by the use of DFs, showing that coding techniques should always be carefully deployed and their advantages well understood. For this reason, we believe this work can provide a useful aid in better appreciating the critical aspects of jointly utilizing multimedia streaming and DF coding techniques for IPTV.

INTRODUCTION

Although commercial IPTV channels are streamed utilizing multicast technologies implemented within network routers, end-to-end peer-to-peer (P2P) streaming techniques received great attention thanks to:

- Their flexibility, as they can easily be put into action at the edges of the Internet with little hardware investments.
- Their efficiency, as they use both the download and the upload bandwidth capabilities of every single peer.

As traditional IPTV services require the installation of networks where all routers support the distribution of multicast flows, P2P streaming can play an important role in heterogeneous network scenarios, where a single provider does not control all of the network equipment (e.g. routers). In such cases, peers can collaborate sharing part of their upload bandwidth to bear the transport of multimedia content requested by other peers. Clearly, such an approach poses the challenge of how the upload bandwidth of every single peer may be used at its best, effectively aiding the process of satisfying a set of stringent performance values defined for IPTV systems.

In particular, three main metrics may be identified to evaluate the performance of an IPTV system:

- Its SETUP delay (i.e. the time elapsed before a channel clearly shows up after a user tunes to it).
- Its END-TO-END delay (i.e. in the case of live streams, for example the broadcast of the soccer world cup final, this represents the delay between when an action, a goal, occurs and the time when it can be effectively enjoyed on the TV).
- Its PLAYBACK continuity (i.e. the ratio between sent and received video frames, as some may be lost or arrive too late at the receiver when traveling on a best effort IP network).

Now, within the domain of P2P streaming strategies, roughly two distribution schemes may be identified that optimize different subsets of the mentioned IPTV performance metrics: trees and meshes. Tree strategies deploy a distribution graph where a server, directly connected to the source of content, pushes the multimedia flow down a tree of peers. When the tree topology is stable, they introduce little overhead traffic and provide good results throughout all performance metrics.

However, tree-based P2P streaming strategies greatly suffer when peers appear and disappear at a high frequency (i.e. high churn). This, in fact, entails destroying and recreating sub-trees of peers with substantial portions of a multime-

This work was partially supported by the Italian FIRB Damasco Project.

0163-6804/12/$25.00 © 2012 IEEE

dia flow potentially lost during this time, thus affecting the PLAYBACK continuity of a flow.

Mesh-based P2P networks, instead, are robust to churn, as each peer is provided, through the periodic exchange of information (i.e. buffer maps) between peers, with the knowledge of which peers own what data, thus enabling it to selectively request the parts that it is still missing. Relying on multiple upstream peers to retrieve multimedia flow packets, such systems are resilient to failures. Nevertheless, this advantage has a cost in terms of the SETUP and the END-TO-END delays a receiver experiences, as it requires that all peers allocate large buffers to support the request of packets.

While P2P streaming techniques have become progressively more sophisticated, a great amount of research has explored new paths, assessing the performance of combining new and innovative digital coding with P2P distribution strategies. Coding techniques can, in fact, provide very promising tools to combat the effects of high churn rates and, in general, of packet losses. While in traditional P2P streaming networks a packet loss directly maps into a loss of information and may be recovered only requiring a retransmission of the missing packet, utilizing Automatic Repeat reQuest (ARQ) protocols, in coded P2P streaming networks the loss of a packet does not necessarily represent a loss of information, as it may be recovered combining other packets that were already (or will be) received. Therefore, using the principle that each packet, sent by each peer, can contribute to receive the entire information sent by a source, coding in P2P streaming can significantly help solve PLAYBACK continuity problems induced by high churn rates, as a loss of one upstream peer does not necessarily translate to a loss of the information flow.

Due to the great interest that has risen around the use of joint coding and streaming techniques, many different approaches have been devised that aim at optimizing the three performance metrics of IPTV systems. The goal of our work is to assess on a real test-bed the true impact of a well-known coding technique, such as Digital Fountains (DFs), when coupled with a popular and robust streaming scheme like BitTorrent. Preliminary results have been anticipated in [1].

Here, instead, the objective of the article is neither the proposal of a new P2P mechanism, nor the invention of a new sophisticated coding strategy, but to verify if the combination of an efficient coding scheme with a stable P2P technique may bring an effective advantage in terms of IPTV distribution. To this aim, we devised BitFountain, a P2P streaming client that integrates a modified version of the popular BitTorrent protocol with the use of DFs. Interestingly, our results obtained testing BitFountain on PlanetLab show that the advantages that may derive from utilizing DFs on BitTorrent-based streaming platforms may only be appreciated as the network scales beyond a certain size.

The remainder of this article is organized as follows. We provide a survey of coding techniques utilized in P2P streaming networks, while we present BitFountain and report on the experimental results that we obtained with our test-bed. We then conclude the article.

Coding Techniques in P2P Streaming Networks

The use of coding techniques in P2P networks has been thoroughly explored in the past few years. We may anyway roughly identify two distinct trends in their use within P2P networks, depending on where the coding activity is performed: either at the source or throughout the network.

A good example within this context is that of Scalable Video Coding (SVC). With SVC a video stream may be divided at the source into multiple layers, with a base layer providing a low quality stream and a set of enhancement layers, which, instead, gradually increase the quality of the video in order to adapt the number of layers sent by a peer to the available bandwidth through its upload channel [2]. An advantage that derives from the use of such a solution is that if the outgoing bandwidth of a peer falls below the value required to send all the layers that compose a stream (e.g. this may happen with asymmetric channels), the peer can decide to only send those layers that, starting from the base layer, can fit into the available bandwidth. However, the hierarchical nature of such an approach requires receiving the layers in a given order, as receiving an enhancement layer, without receiving the base layer, would result in being unable to decode the video content. Multiple Description Coding (MDC) removes the hierarchical organization of SVC (i.e. there is no distinction between layers). A source sends descriptions (instead of layers) that, once received, can always be decoded. Briefly, we may imagine receiving a single description as receiving the base layer of an SVC-based stream. However, its advantage is that receiving richer subsets of descriptions produces a better quality of service as more descriptions reach the receiver. A P2P streaming system that puts to good use MDC is SplitStream [3]. Such a system relies on a forest topology (i.e. multiple trees) where a peer can be a leaf in all but one of the trees. By sending a single description through each tree, if any peer part of the SplitStream network fails, this event will require the recovery of only one tree, while all the other peers will seamlessly keep streaming their video descriptions through the other trees that compose the forest.

Mesh-based systems like the popular BitTorrent, instead, have experienced widespread use of network coding techniques, where coding operations are implemented not only at the source of content (e.g. video server), but also within each peer that cooperates in a P2P network. The purpose of introducing coding operations at all peers may be briefly explained as follows. In traditional file exchange systems, a peer that is missing one block of data will finish its download only after that block is found and retrieved from some peer. Using network coding, instead, a peer only needs to find a block that is independent from all the blocks that have been received before, as a block with such property carries new information and is useful to complete the decoding process. One of the first P2P clients that has supported network coding is Avalanche, although its application is mainly focused on the scenario of bulk data downloads for file sharing [4]. In brief, each Avalanche peer generates and sends blocks that

Mesh-based systems like the popular BitTorrent have experienced widespread use of network coding techniques, where coding operations are implemented not only at the source of content but also within each peer that cooperates in a P2P network.

The DFs approach has large implications on the structure of the communication protocol: the need for feedback from the receivers is drastically reduced, with significant advantages that can be observed on the complexity and scalability of the protocol and on its applications.

result from a linear combination (i.e. a XOR) of blocks of a given file, while each receiver reconstructs the original file when a sufficient number of linearly independent combinations of blocks are received. Such an approach has also been applied to the specific context of P2P streaming networks, where, for example, another protocol, named R2, randomly pushes blocks of data, achieving a reduced SETUP delay and an improved PLAYBACK continuity [5].

Of particular interest for our study are Digital Fountains (DFs). While at the beginning they were primarily thought of as source codes, they have been later employed also as network codes. In essence, DFs rely upon one simple principle: a generic file at a sender can be first divided into a number of subparts, say k, and then be encoded into an infinite number of encoded symbols. Once a sufficient number, possibly k (or slightly more), of the encoded symbols created by the sender reach a receiver, the whole file can be ideally rebuilt very rapidly and the sender can stop transmitting any more data.

Erasure Codes, which can be used to provide a first approximation of the DF principle, have been utilized to efficiently transfer bulk data from a server to a set of peers. However, the set of encoded symbols received by a generic peer can significantly coincide, posing an issue well known in BitTorrent as the reconciliation problem (e.g. finding a missing symbol/block within a mesh-based P2P system).

To overcome this problem, the authors of [6] proposed a system that requires the dissemination of additional information about the encoded data stored at each peer, in a fashion that is similar to the exchange of buffer maps between peers in a mesh-based topology. However, more advanced approximations of DFs, namely rateless codes, produce potentially infinite sequences of uniquely encoded symbols which eliminates any overlap, thus providing a solution to the reconciliation problem that avoids the exchange of any supplemental information. Such a solution has been effectively exploited decoding, re-encoding and sending newly encoded data within all peers, thus implementing a network coding approach that explores the diversity introduced by each participating peer (i.e. producing a greater number of independent encoded symbols) [7, 8]. Reed Solomon, Tornado, Luby Transform, and Raptor codes all represent, respectively, increasingly more successful approximations of the DF paradigm and have been tested on P2P streaming platforms [9, 10].

In the following subsection we are going to provide an example that will indicate the possible advantages of utilizing DFs, instead of ARQ mechanisms, in the streaming of multimedia content.

ARQ vs. Encoding Techniques

The DFs approach is based on a redundant encoding of data, which makes the protocol tolerant to packet losses and client heterogeneity. Since each received packet conveys information on a (potentially large) number of data blocks, reception of specific blocks is not necessary; rather, it is the total number of received blocks that determines the successful completion of the data transfer. This approach has large implications on the structure of the communication protocol: the need for feedback from the receivers is drastically reduced, with significant advantages that can be observed on the complexity and scalability of the protocol and on its applications.

To illustrate why the use of coding could be beneficial in the context of P2P streaming networks, consider a scenario where a video is split into three blocks (x, y, z). The P2P network is composed of seven peers. Six peers, namely A, B, C, D, E, and F, contain all three blocks each. One peer, named J, misses all blocks. Assume J is aware of A, B, C, D, E, and F and proceeds making the following requests: x to A and B, y to C and D, while z to E and F (part (a) of Fig. 1). Furthermore, let us also assume that only 50 percent of the contacted peers will answer to J's requests. The following scenario might occur. Peers A and B answer, so J receives two exact copies of block x, thus wasting part of its download bandwidth. Peer C answers, while D does not, thus one copy of block y becomes available. Finally, E and F do not answer, therefore J is unable to retrieve a copy of block z. To complete the download process, J needs to issue a further request to receive block z. This solution exposes one drawback of the ARQ paradigm: longer delays and greater overheads are induced by retransmission requests. Redundant requests devised to alleviate such effects, on the other hand, can produce the side effect of wasting bandwidth resources (e.g. J received one unnecessary copy of x). Now, let us see what would have happened with this same example when using a basic DF scheme. Implementing a DF scheme at each peer, all symbols transferred between two peers result from the XOR operation between the blocks that compose the file. Moreover, using a random Digital Fountain scheme, for example, a P2P client sends the XOR of blocks that are randomly picked among those that compose a file. So, applying a simple approximation of a DF, a forward error correction (FEC) mechanism, the following could occur (part (b) of Fig. 1). Peer A answers sending M, which is equal to (x XOR y). B, instead, answers $N = x$ XOR y XOR z. Finally, C answers with O, which equals (y XOR z). As in the previous case, only 50 percent of the peers that have received block requests have answered to J, but J is now able to decode the three encoded blocks obtaining the original file. In fact, x is equal to (N XOR O), y is equal to (M XOR N XOR O), while z equals (M XOR N). Using a simple DF in this case, there is no need to ask for a retransmission of a missing block.

Now, together with the advantages, we need to know which are the hurdles introduced by DFs in a communication system as an IPTV platform. In general, two parameters evaluate the efficiency of a coding technique:

- Overhead: if k is the length of stream in blocks and N is the total number of blocks sent to reassemble it, $\varepsilon\, k = N - k$, with $\varepsilon > 0$, gives us how many more blocks have been sent between a sender and a receiver compared to the initial file size k.
- Complexity: the total number of operations performed among blocks during the coding process to produce one encoded symbol and during the decoding process to recover the original data.

In Table 1, as drawn from [4, 9], we summarize the performances of linear network coding with the performances of the most well know DF coding techniques. As we may see, Raptor codes today provide among the best result in terms of both overhead and computational complexity, being the ones that today best fit into the DFs paradigm. For this reason we decided to test such DF flavor within a BitTorrent platform, an experience that we will briefly describe in the next section.

BitFountain

We report on BitFountain, a BitTorrent client that uses DFs for encoding and decoding activities. The reason for choosing a mainline BitTorrent client, rather than redesigning it, was motivated by the fact that we aimed at assessing the added value provided by DFs with respect to a standard P2P client [11]. In the following, we are going to illustrate the steps we followed in designing our BitFountain client, along with some experimental results obtained from a widespread testing campaign performed on the PlanetLab platform [12].

At the transport layer, the difference between a pure BitTorrent and our BitFountain is that the former transfers data utilizing reliable TCP connections, while the latter utilizes unreliable UDP segments. Such a decision follows from the general design choice of dealing with the loss of any data with coding techniques, rather than with ARQ. Indeed, a first modification we implemented with respect to BitTorrent was that of compelling our clients to behave always as if they were in endgame mode (i.e. while looking for rare blocks). In other words, this entails that each peer, searching for a block, always asks it to the entire community of its un-choked neighbors (that is, peers that may answer to that request), as all blocks with DFs deserve the same treatment.

A second major design decision has been that of choosing the level where DFs should be integrated within a BitTorrent client. To this aim, we should recall that three major data levels exist in BitTorrent: files, pieces, and blocks. A file is composed of multiple pieces, which, in turn, are each composed of multiple blocks. The default block size is 16 kB, while the piece size varies.

Hence, if we had decided to apply a DF encoding method to BitTorrent pieces we would have had the following unpleasant situation. As the input file is divided into pieces, at every incoming request, a DF encoder would randomly select which pieces to encode. After this step, that given random piece would be sliced accordingly to the length specified in the request, obtaining a random block, to be sent to the requester. A major problem arises with the aforementioned technique due to the fact that during the coding activity all the encoded pieces are necessary to reconstruct the file.

This is an acceptable factor for downloading, but not for streaming. When streaming, indeed, we would like to have each single piece already decoded and available to be streamed. The solution to this problem is to move the encoding activity at the block level. In this way, a piece can be streamed as soon as all of its encoded blocks are available.

In brief, when a peer receives a request for a block inside a piece, it randomly selects a subset of blocks within that piece and it XORs them, obtaining an encoded block that is hastily sent. When the peer that originated the request receives a block belonging to a given piece, it tries to decode it using the encoded blocks received so far. If the decoding process fails, the peer issues requests for new encoded blocks until it can decode the original piece.

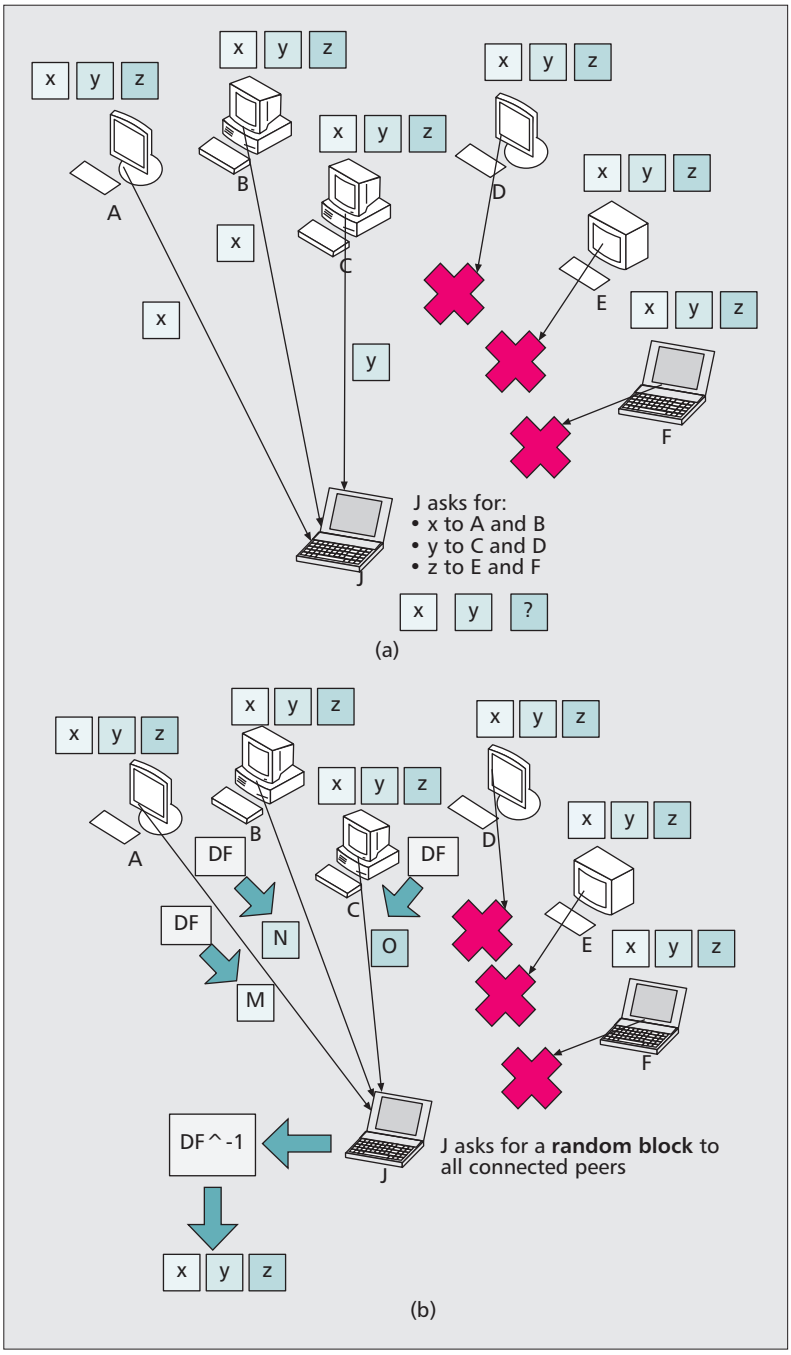

Figure 1. *ARQ vs. FEC.*

Results and Discussion

We report on some results that were gathered from a testing campaign performed on PlanetLab with: 170 peers, one single multimedia server, and a maximum upload speed of 1.6 Mb/s. Figure 2 shows the download times of a file of 30 MB divided into pieces of 2 MB.

Coding Technique	Overhead	Coding and Decoding Complexity
Linear Network Coding	None	O(k), O(k^3)
Reed Solomon	None	$O(k)$, $O(k^2)$
Tornado	$\varepsilon\, k$	$O(N \ln(1/\varepsilon))$, O(N ln (1/ε))
Luby Transform	$\propto k$	$O(\ln N)$, $O(N \ln N)$
Raptor	$\varepsilon\, k$	$O(\ln(1/\varepsilon))$, $O(N \ln(1/\varepsilon))$

Table 1. *Coding techniques properties.*

	Avg. coding time [s]	Avg. delivery delay [s]	Idle time (1–170 peers) [s]
BitFountain (no pre-XOR)	30	150	[205–230]
BitTorrent	0	150	[200–235]

Table 2. *Download time components.*

Architecture	CPU frequency, memory	Avg. encoding/decoding time [ms]
Intel QuadCore (PlanetLab node)	3 Ghz, 4 GB	3.9, 6.3
Intel Core Duo	2.5 Ghz, 2 GB	6.5, 8.7
Intel Pentium IV	3 Ghz, 2.6 GB	10.3, 17.9

Table 3. *Coding times.*

As shown in Fig. 2, we tested four clients:

- BitTorrent.
- BitFountain.
- A BitFountain client where all the coding operations are pre-computed (tagged with pre-XOR in Fig. 2).
- A BitFountain TCP client that utilizes TCP at the transport layer and where, again, all coding operations are pre-computed

Obviously, the download times were chosen as the major figure of merit, as they represent a clear indicator of how well an IPTV system may work.

With this in view, the most evident result is that standard BitTorrent outperforms BitFountain with no pre-XOR; this happens for the simple reason that BitFountain (with no pre-XOR) clients waste about 30 seconds on average in the coding process, as also highlighted in Table 2, where the download times were broken into the three relevant delays that files experience before a download is completed (coding, delivery, idle time waiting for other peers' un-choking, as a function of the peer number).

Nonetheless, if we consider the BitFountain clients where coding is pre-computed, these peers achieve more or less the same performance as their BitTorrent counterparts (as shown by the dotted curve in Fig. 2).

As to the relationship between the time wasted during the coding activities and the computation power of the machines on which such activities were carried out, it is interesting to note the results presented in Table 3. That table reports on the average coding times, required on a typical PlanetLab node, over 500 experiments performed with our Python implementation, on a single block of 16 kB. These results were contrasted with the times taken on other machines (of lower capabilities), thus witnessing the importance of high processing power in such phases.

A more careful analysis of Fig. 2 shows that the completion time of BitTorrent clients steeply increases between peers 100 and 130, thus showing that when the network increases beyond a given population size, BitTorrent experiences a performance degradation.

In fact, the larger the BitTorrent network size, the smaller the probability of finding an un-choked peer able to deliver to the requesting client the block of interest through a small number of intermediaries. Instead, with BitFountain, the probability that an un-choked peer owns an encoded block increases with the number of peers taking part in the encoding activity.

This is one of the positive effects of network coding, which becomes more evident with large communities of encoding nodes, and also explains why with a population of around 120 peers BitFountain(with pre-XOR) outperforms BitTorrent. It is also interesting to note that Fig. 2 is only capable of capturing the beginning of a trend where BitFountain improves stably over the performance of BitTorrent. This fact is encouraging, as the expected size of an IPTV network is in the order of thousands and even millions of customers, and should be kept in mind when designing future platforms.

The last result of a certain interest reported in Fig. 2 concerns the idea of utilizing TCP as the transport layer for BitFountain (with pre-XOR). Figure 2 clearly demonstrates that this solution is not the winner as the interferences between the ARQ mechanism of TCP and the redundancy provided by BitFountain is paid in terms of larger completion delays.

Finally, we decided to verify the effect of varying the piece size on the two protocols (BitFountain with pre-XOR vs. BitTorrent) while keeping the default block size (16 kB) fixed.

Figure 3 shows that increasing the piece size from a value of 64 kB to a value of 8 MB increases the completion time of both schemes, although BitFountain suffers this test less than BitTorrent.

Conclusion

We analyzed the progress made by research in the field of using coding and P2P streaming strategies for IPTV. Our experimental results show that this strategy brings interesting advantages, as soon as the IPTV network reaches a certain size.

References

[1] G. Marfia *et al.*, "Digital Fountains + P2P for Future IPTV Platforms: A Test-bed Evaluation," *Proc. 4th IEEE/IFIP Int'l. Conf. New Technologies*, Mobility and Security, Paris, 2011, pp. 1–5.

[2] P. Baccichet *et al.*, "Low-Delay Peer-to-Peer Streaming Using Scalable Video Coding," *Proc. 16th Packet Video Wksp.*, Lausanne, 2007, pp. 173–81.

[3] M. Castro *et al.*, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," *Proc. 19th ACM Symp. Operating Systems Principles*, Bolton Landing, 2003, pp. 298–313.
[4] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," *Proc. 24th IEEE Annual Joint Conf. IEEE Computer and Commun. Societies*, Miami, 2005, pp. 2235–45.
[5] W. Mea and B. Li, "R2: Random Push with Random Network Coding in Live Peer-To-Peer Streaming," *IEEE JSAC*, vol. 25, no. 9, Dec. 2007, pp. 1655–66.
[6] D. Kostic *et al.*, "Bullet: High Bandwidth Data Dissemination Using An Overlay Mesh," *Proc. 19th SIGOPS Oper. Syst. Rev.*, Bolton Landing, 2003, pp. 282–97.
[7] W. Chuan and B. Li, "rStream: Resilient and Optimal Peer-to-Peer Streaming with Rateless Codes," *IEEE Trans. Parallel Distrib. Sys.*, vol.19, no.1, Jan. 2008, pp. 77–92.
[8] N. Thomos and P. Frossard, "Collaborative Video Streaming with Raptor Network Coding," *Proc. IEEE 2010 Int'l. Conf. Multimedia and Expo*, Hannover, 2008, pp. 497–500.
[9] M. Mitzenmacher, "Digital Fountains: A Survey and Look Forward," *IEEE Info. Theory Wksp.*, 2004, 24–29 Oct. 2004, pp. 271–76.
[10] S. Spoto *et al.*, "BitTorrent and Fountain Codes: Friends or Foes?," *Proc. IEEE Int'l. Symp. Parallel & Distributed Processing*, Atlanta, 2010, pp.1-8.
[11] B. Cohen, "Incentives Build Robustness into BitTorrent," *Proc. 1st Economics of Peer-to-Peer Systems Wksp.*, Berkeley, 2003.
[12] B. Chun *et al.*, "Planetlab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 33, no. 3, July 2003, pp. 3–12.

Biographies

Alessandro Cattaneo (alessandro.cattaneo@st.com) is a Research Intern with STMicroelectronics, Agrate Brianza, Italy. He received a Laurea Degree in Computer Engineering in 2010 from the Politecnico di Milano, Italy.

Gustavo Marfia [M'11] (marfia@cs.unibo.it) received a Laurea degree in Telecommunications Engineering from the University of Pisa in 2003 and a Ph.D. in Computer Science from the University of California, Los Angeles, in 2009. He is currently a Postdoctoral Researcher at the Department of Computer Science of the University of Bologna. His research interest include biomedical, transportation and multimedia networking systems, fields in which he has published over 40 technical papers.

Alexandro Sentinelli (alexandro.sentinelli@st.com) received a Laurea degree in electrical and Telecommunications Engineering from the Università di Roma 3 in 2003. From 2005 to 2007 he has been a researcher for STMicroelectronics at the University of California, Los Angeles (UCLA). In 2003 he joined the Research and Development Laboratories of Motorola Italia, Turin, Italy. Since 2007 he is with STMicroelectronics AST Labs in Agrate Brianza, Italy. His research interests are physical layer, information theory, and overlay networks in P2P environments for video streaming applications.

Andrea Vitali (andrea.vitali@st.com) is a Senior Engineer at STMicroelectronics, Agrate Brianza, Italy. He received a Laurea Degree in Electrical Engineering from the Politecnico di Milano, Italy. His research interests include digital signal processing, video/audio processing, source/channel coding and human-computer interfaces design.

Luca Celetto (luca.celetto@st.com) is a Software Design Manager with STMicroelectronics, Agrate Brianza, Italy, where he manages a team of R&D employees and consultants on streaming and information retrieval, exploring innovative software solutions and architecture bottlenecks. He received a Laurea Degree in Microelectronics Engineering from the University of Padua, Italy.

Marco Roccetti (roccetti@cs.unibo.it) received the Italian Laurea degree in Electronic Engineering from the University of Bologna, Bologna, Italy. He has been a Full Professor with the Computer Science Department, University of Bologna, since 2000. His research interests include computer entertainment, intelligent transportation systems and peer-to-peer applications, fields in which he has authored more than 200 technical papers. He serves as an Associate Editor for many international journals and is active in several Italian and international projects.

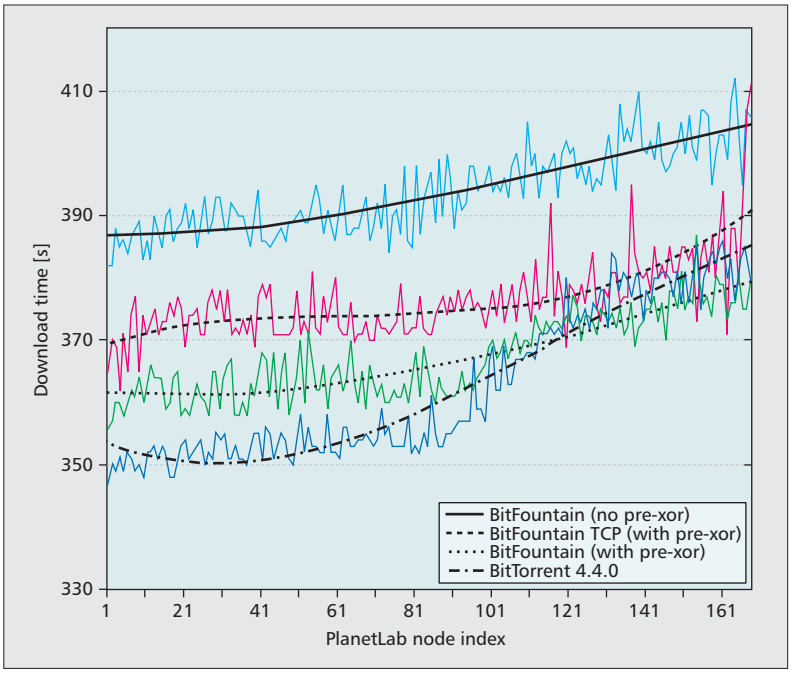

Figure 2. *BitFountain vs. BiTorrent: download times with and without pre-coding.*

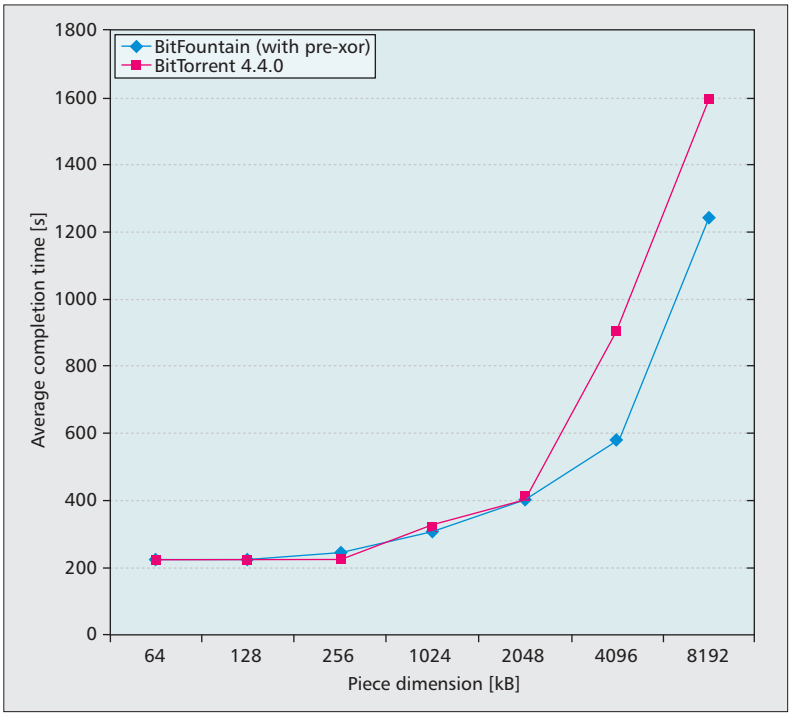

Figure 3. *BitFountain (with pre-XOR) vs. BiTorrent: download times as a function of the piece size.*

Mario Gerla [F'02] (gerla@cs.ucla.edu) received a graduate degree in engineering from the Politecnico di Milano in 1966, and his M.S. and Ph.D. degrees in engineering from UCLA in 1970 and 1973. He joined the Faculty of the Computer Science Department at UCLA, where he is now a professor. His research interests include distributed computer communication systems and wireless networks. Currently, he is leading the ONR MINUTEMAN project at UCLA, with a focus on robust scalable network architectures for unmanned intelligent agents in defense and homeland security scenarios. He also conducts research on scalable TCP transport for the next-generation Internet.