TOWARDS A P2P VIDEOCONFERENCING SYSTEM BASED ON LOW-DELAY NETWORK CODING

Attilio Fiandrotti, Anooq Muzaffar Sheikh, Enrico Magli

Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino
attilio.fiandrotti@polito.it, anooq.sheikh@polito.it, enrico.magli@polito.it

ABSTRACT

We present ToroStream, our P2P video streaming protocol designed around Network Coding (NC), and we investigate its applicability to low-delay applications such as live video streaming and videoconferencing. While NC offers several benefits for P2P content distribution, the coding operations at the network nodes introduce a delay that is detrimental for low-delay communications. To this end, we designed a push-based packet scheduling mechanism devised to minimize the communication delay. Furthermore, our protocol is designed around a low computational complexity class of rateless codes that enable us to experiment on real networks where the nodes have limited computational capabilities. That enables us to experiment with two typical low-delay applications such as live video streaming and videoconferencing in a realistic network scenario. The experiments show that our protocol enables continuous playback with limited initial delay and we point out the issues that NC poses in low-delay applications.

Index Terms— P2P, Network Coding, Low-Delay

1. INTRODUCTION

Network Coding (NC) [1] has emerged as an interesting solution for P2P communications. We consider a network scenario where one source node wants to distribute a message to multiple cooperating receivers, the *peers*. The source node divides the message in blocks of equal size, for example the size of a network packet, and transmits linear combinations of such blocks to the peers. The peers store the encoded packets and exchange linear combinations of the received packets among themselves. Once a peer has collected enough linearly independent packets, it solves the related system of linear equations and decodes the message. The recombinations at the nodes, which are the distinctive characteristic of NC, improve the performance of P2P media distribution architectures in many ways. First, all the packets are equivalent to the end of recovering the original message: that solves the issue of the asymmetric popularity of network packets, also known as the rarest-piece issue. Second, the recombinations reduce the probability to transmit redundant information even if loops are present in the network. That makes possible to organize the nodes into an unstructured overlay, which makes the communication more resilient to network failures and peer churning. Third, the network nodes do not need to explicitly request specific parts of the message, which saves time and bandwidth (*push*-based packet scheduling.) Finally, the peers can start to transmit packets before they have recovered the message, which is a major edge in delay-sensitive applications such as video streaming. The benefits of NC for P2P media distribution have been highlighted before. For example, Wang and Li [2] described a P2P video streaming protocol designed around a push packet scheduling mechanism, showing that NC increases the streaming throughput and reduces the initial buffering delay, improving the quality of experience of the user. Our experiments [3] with a push-based video streaming architecture based on multiple trees showed reduced buffering times together with improved resilience to ungraceful peers departures.

The coding operations at the network nodes introduce however a detrimental delay for videoconferencing applications and the computational complexity of such operations prevents to assess the impact of such delay on the user's quality of experience in real networks where the nodes have limited computational capabilities. Due to the computational requirements of NC, most of the existing research in the field of NC-based communications relies on simulations over synthetic testbeds. For example, the authors of [2] experimented on a cluster of high performance servers connected on a local area network that provided the required computational power. While such experiments suggest that NC enables reduced buffering times, it is not totally clear whether such conclusion holds and to which extent in real networks where the packets are affected by erasures, delays and out of order delivery.

In this paper we present *ToroStream*, our P2P protocol for live video streaming designed around NC. Our protocol is devised to minimize the communication delay thanks to a push-based packet scheduler and an efficient feedback mechanism. The protocol is designed around a family of low-complexity rateless codes that we introduced in [4], which enables us to experiment on a real network where the nodes have limited computational capabilities. We perform live video streaming experiments and we evaluate the quality of the video delivered to the users as a function of initial buffering time and the bandwidth contributed by the source node. Our experiments show that our protocol enables continuous playback with little startup delay even on lossy networks. Then, we explore a videoconferencing application showing that, while NC does allow for low-delay communications, there are issues that need to be addressed in order to efficiently use the network resources.

2. THE TOROSTREAM PROTOCOL

2.1. Network Coding with Rateless Codes

In this section, we overview the basic principles of NC with rateless codes. The source node holds a video content, which is subdivided in chunks of data called *generations*. The source divides each generation into a sequence $(M_0, ..., M_{N-1})$ of N data blocks of identical size, where N is called *generation size*. The source transmits linear combinations of the blocks of a generation as $X = \sum_{i=0}^{N-1} g_i M_i$, where the sum operator is the bit-wise XOR. The vector $g = (g_0, ..., g_{N-1})$ is called *encoding vector* and $g_i \in GF(2)$ is drawn so that $\mathbb{P}\{g_i = 1\} = \frac{1}{2}$. The source transmits the packet $P(g, X)$ that contains the encoded payload X plus the corresponding encoding vector g so that the packet can be decoded at the receiver node [5].
The peer nodes receive encoded packets and transmit linear combinations of the received packets. Every time a transmission opportunity arises for a node, the node selects some of the received equations and recombines them into a new packet which is transmitted to the network. Packet recombination serves the fundamental purpose of increasing the likelihood that a packet transmitted by a node is linearly independent from all the packets previously collected by the receiver node. If a packet is linearly independent from all the previously received packets, the packet is said to be *innovative*.
The peer nodes receive encoded packets and recover the corresponding generation solving a system of linear equations. Once a node has collected N innovative packets, the node solves the corresponding system of linear equations with Gaussian Elimination and recovers the original message. In the ideal case, all the packets received by a node are innovative and the generation is recovered after N packets have been received. In practice, not all the received packets are innovative due to the random encoding process at the source and the random recombinations at the peers. Therefore, it is usually necessary to receive $N' > N$ packets to recover the generation. Assuming that a node receives packets at a constant rate, the time required to decode a generation, and thus the communication delay, decrease if the received packets are likely to be innovative.

2.2. Peer Discovery and Overlay Management

The discovery of the nodes in the network and their organization into an unstructured overlay of peers is managed by a central *tracker*. The tracker maintains a list of all the peers in the overlay and listens for join requests. A node that wants to join the overlay contacts the tracker and the tracker adds the node to the list. The tracker replies to the node with a list of randomly selected addresses of nodes that are already in the overlay. The node starts a separate handshake procedure with each address found in the list, upon which the two nodes become peers. Once two nodes are peers, they start to exchange video packets without further delays thanks to the embedded feedback mechanism described in the following. Peers periodically exchange keepalive messages to detect network failures and avoid transmitting packets to nodes that have become unreachable. If a node does not receive any keepalive from a peer for a given amount of time, that node is removed from the list of the peers and no mode packets are transmitted to it.

2.3. Feedback with Embedded Decoding Vectors

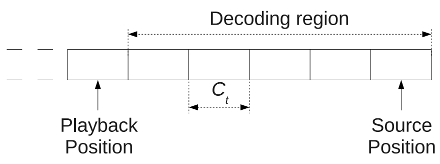

Fig. 1. The video stream as a sequence of generations.

Figure 1 shows a video sequence subdivided in several generations of data. A generation corresponds to one or multiple self-decodable units of video such as Groups of Pictures (GoPs.) For the sake of simplicity, in the following we assume that the video is encoded at constant bitrate, thus all generations have approximately equal size N and identical playback duration C_t. Every C_t seconds, the source node fetches one generation from the video source (e.g.: a pre-encoded sequence.) For the following C_t seconds, the source encodes and distributes packets only for that generation, which is defined as the *source position* in the video stream. Before starting the playback, a node buffers $t_b \geq C_t$ seconds of video. After t_b seconds of buffering, the node attempts to play the first generation of video. If the generation is not decoded prior to its playback deadline, the screen freezes for the duration of the missing generation and the quality of the video experience degrades. Every C_t seconds, the peer updates its playback position, attempts to play another generation and so on for the rest of the streaming session.

The set of generations encompassed between the position of the source in the video stream and the playback position of a node is known as the *decoding region* of the node. The array of binary variables that describe the status (decoded or not decoded) of the generations within the decoding region of a node is defined *decoding vector*. The decoding region usually corresponds to a few generations of video, so the relative decoding vector can be represented with few bits only.

For example, if the buffering time is equal to five seconds (i.e., $t_b = 5s$) and each generation of video corresponds to 1 second of video ($C_t = 1s$), the encoding vector can be represented with exactly five bits. The reduced size of the encoding vector makes possible to embed it in each transmitted packet with a negligible increase in the signaling overhead. For example, in the case of an Ethernet network where the frame size is 1500 bytes, the size of an encoded packet would be about 1000 bytes while the encoding overhead would be just 5 bits, which yields to an overhead of $\sim 10^{-3}$. Specifically, a decoding vector is embedded in the message that two nodes exchange during the handshake, which enables the nodes to start transmitting video packets right after the handshake without the need to exchange buffermaps. The decoding vector received during the handshake is stored and exploited at the moment of transmitting a packet as explained in the following. Every time a node decodes a generation, it sends an appropriate *stop* message to all its peers, which update the stored decoding vector relative to the node. However, stop messages could be lost and the peers of a node may miss important updates about the decoding vectors of its peers. To this end, a decoding vector is embedded in each packet that a node transits to its peers. The received decoding vectors are exploited by the peers to compute appropriate packet scheduling policies, as explained in the following.

2.4. Push-Based Packet Scheduling

Each node periodically transmits a packet and is up to the packet scheduler to decide which peer to address and for which generation to transmit the packet. Each time a transmission opportunity arises for a node, the packet scheduling Algorithm 1 is executed. A round-robin policy is enforced to guarantee that the limited output bandwidth of the node is fairly allocated among its peers. Then, the scheduler checks the last decoding vector received by the peer. If the peer has already decoded all the generations in its decoding region, another peer is selected in round-robin fashion. Otherwise, the generation that is closer to the playback deadline is selected for transmission to account for a urgency principle. At this point, the scheduler recombines some of the received equations for the selected generation, transmits the packet and waits for the next transmission opportunity.

Algorithm 1 Round-Robin Packet Scheduler.

```
1: for each peer $\mathcal{P}^i$ do
2:     check last decoding vector received by $\mathcal{P}^i$
3:     if $\mathcal{P}^i$ has decoded all generations
4:         select peer $\mathcal{P}^{i+1}$ and go to line 2
5:     select generation closest to playback deadline
6:     recombine received packets for the generation
7:     transmit the recombined packet
8:     break
9: end for
```

3. EXPERIMENTAL EVALUATION

In this section, we experiment with our P2P video streaming protocol in two different network scenarios. First, we consider a controlled conditions testbed composed of several workstations connected via gigabit switch that provides the bandwidth necessary to experiment with a large number of nodes in error free network conditions. Then, we experiment on PlanetLab [6], a network composed of hundreds of Internet nodes that enables us to experiment with packet erasures, delays and out-of-order delivery. For each network scenario, we consider two different low-delay applications. First, we consider a live video streaming application where the maximum delay the user can endure corresponds to a few seconds of video buffering. Then, we consider a videoconferencing application where the delay constraints are even tighter and are well below one second. We evaluate the quality of the video perceived by the user as a function of various combinations of protocol parameters.

3.1. Controlled Conditions Simulations

First, we consider a live video streaming application where one source node distributes a live video stream to hundreds of users. We investigate the relationship between buffering time and quality of the video perceived by the user. The buffering time corresponds to the initial delay that the user has to endure before the playback starts. From the user's perspective, it is desirable that the initial delay that has to be endured is as small as possible. On the other hand, a high buffering time reduces the likelihood that the screen freezes because a generation could not be decoded ahead of its playback deadline. The source node streams a 10 minutes long H.264/AVC video sequence encoded at $B_v = 500$ kbit/s which is subdivided in generations of 1 second each (i.e., $C_t = 1s$.) The network packets are 1250 bytes each, which yields to generations composed by $N = 50$ packets. We experiment with a number of peer nodes N_p that ranges from 100 to 300 to evaluate the ability of our protocol so cope with large the population sizes. The output bandwidth B_s of the source node is dependent on the number of nodes in the network and is constrained so that each peer receives no more than 10% of the packets from the source. The output bandwidth of the peer nodes B_p is equal to the encoding bandwidth of the video ($B_p = B_v$). This setup guarantees that the average bandwidth $B_{in} = \frac{B_s}{N_p} + B_v$ in input to a peer node is higher than the encoding bandwidth of the video (i.e., $B_{in} \geq B_v$,) so that a node is able to collect the packets necessary to rebuild a generation in less that C_t. The quality of the video delivered to a peer node is measured in terms of Continuity Index (CI), which is defined as the fraction of generations that could be decoded prior to the playback deadline. The effect of the buffering time on the CI is evaluated considering different buffering times $t_b \in [1, 2, 3, 4, 5]$ seconds range. Figure 2 shows the average CI measured at the peer nodes for different N_p and t_b. The figure shows that, for $N_p = 100$, 1 second of buffering is enough to achieve a CI equal to one. As the population of peers increases in number, the average propagation delay between the source and the peer nodes increases as well. Therefore, as the number of peers in the overlay increases to 200 and 300, the buffering time required to achieve a CI equal to one increases to 2 and 3 seconds respectively. These simulations show that out P2P protocol enables almost continuous video playback with limited initial buffering time. The lower bound to the buffering time seems to be related to the average delay required by a peer to collect the packets required to decode a generation, which depends on the link delay between the peers and the number of nodes in the network.

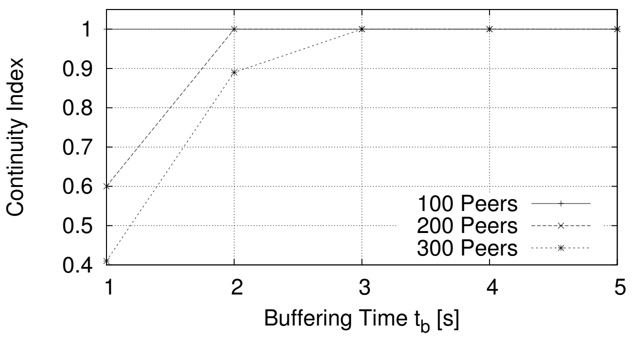

Fig. 2. Average video quality as a function of buffering time and number of nodes in the network (live streaming.)

Then, we consider a video conferencing application where one source node shares a video stream with 10 receiver peers. In such application, the maximum end-to-end delay tolerable by the user to achieve quasi-realtime communications is below half second. To this end, we fix the buffering time to be equal to the generation time (i.e., $t_b = C_t$) and we experiment with $C_t \in [0.5, 0.25, 0.125]$ seconds, which yields to generation sizes of $N \in [25, 13, 7]$ respectively. Furthermore, we gradually increase the output bandwidth of the source node from 10% to 40 % of the total upload bandwidth available in the network. Figure 3 shows the average CI at the peers as a function of the bandwidth contributed by the source node and the size of the generation, where the latter determines the end-to-end delay.

The figure shows that our P2P protocol based on NC enables almost uninterrupted video playback even when the delay is below 0.5 s. However, the figure also shows that, as C_t decreases, more upload bandwidth is required to recover a generation. In fact, the probability that a node is able to decode a generation after receiving N packets decreases as N decreases. Therefore, more upload bandwidth is required to achieve a CI close to 1 as the generation size N decreases. Figure 4 shows the minimum bandwidth that the source node must contribute to each peer with respect to the video bandwidth to achieve a CI equal to one. The figure shows that,

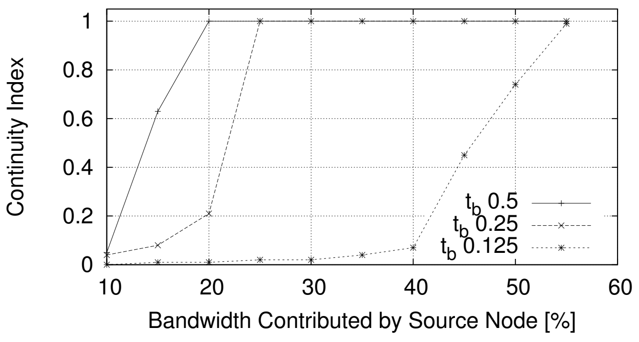

Fig. 3. Average video quality as a function of buffering time and source bandwidth (videoconferencing application.)

for $C_t = 0.125s$, the source node must provide one packet out of two to each peer node in the network. Thus, while this experiment shows that our P2P protocol is suitable for nearly realtime video communications provided that the source node must be able to sustain an increased output bandwidth.

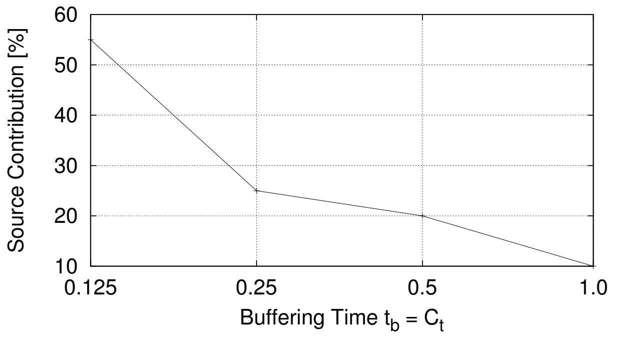

Fig. 4. Mininum bandwidth contributed by source node for continuous playback (CI=1) in videoconferencing scenario.

3.2. Planetlab Simulations

First, we consider a video streaming application where the nodes can tolerate no more than five seconds of video buffering. As for the experiments shown in Figure 2, we stream a video sequence encoded at $B_v = 500$ kbps which is subdivided in generations of $C_t = 1s$. The testbed consists of 150 randomly selected PlanetLab nodes that host the peers, plus one node located at the Politechnic of Turin that hosts the source. Due to the physical constraints on the output bandwidth available at the source node, we could not match the 10% of packets provided by the source of the testbed experiments. Therefore, the experiments are performed for a source node output bandwidth B_s equal to 5 and 2.5 Mbps, which correspond to 6.6% and 3.3% of the packets necessary to each peer to recover a generation. To compensate the reduced output bandwidth of the source, we increased the buffering time t_b to 5 s. Figure 5 shows the results of the experiments (the average CI is equal to 0.97 for $B_s = 5Mbps$ and 0.96 for

$B_s = 2.5 Mbps$.) Most nodes are able to recover the whole video sequence, some exhibit a CI lower that 90% and a few could not decode any generation. The analysis of the log files showed that the input links of these latter nodes are affected by high packet loss rates, which results in insufficient received packets to decode the generations. Due to the round robin packet scheduling policy implemented by our protocol, the same amount of packets is transmitted to every node regardless the actual packet loss probability that affects its input link. This experiment suggests that accounting for the loss probability of the input link of every node could improve the performance of the protocol. Nevertheless, this experiment also suggests that it is possible to serve more than 100 nodes with an average CI above 90% and the source note providing just 5% of the packets necessary to recover the video.

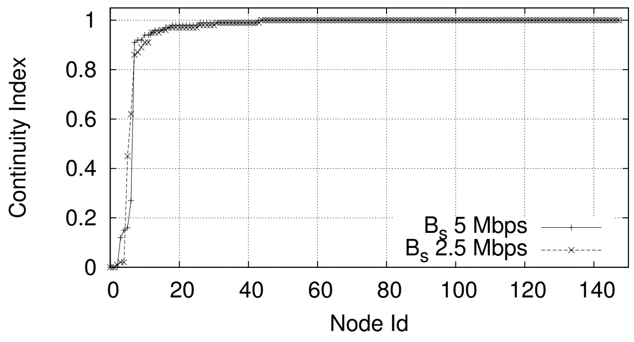

Fig. 5. Video quality as a function of source bandwidth.

Then, we consider a video conferencing application with a reduced number of nodes $N_p = 20$. As for the experiments reported in Figure 3, we consider C_t values equal to 0.25, 0.5 and 1 second and t_b values multiple of C_t. However, as we have previously shown in Figure 4, the bandwidth required to decode the video soars as C_t decreases. the output bandwidth of the peer nodes B_p is increased above B_v to evaluate whether the extra bandwidth could be provided by the peer nodes rather than by the source, whereas B_s is fixed to 2 Mbps (i.e., the source contributes at most 20% of the packets.) Figure 6 shows the results of the simulations. A CI above 95% could be achieved for $t_b = 250ms$, however a CI close to 1 is achieved only for $t_b \geq 0.5s$.

4. CONCLUSIONS

We presented our P2P protocol for video streaming designed around NC and evaluated its performance both on a controlled conditions testbed and on the realistic PlanetLab testbed. The experiments with a live video streaming application showed that our protocol enables to support hundreds of users with a small initial buffering. The experiments with a videoconferencing application also showed that NC-based P2P video streaming is suitable for low-delay video communications. However, our experiments also highlighted that low-delay

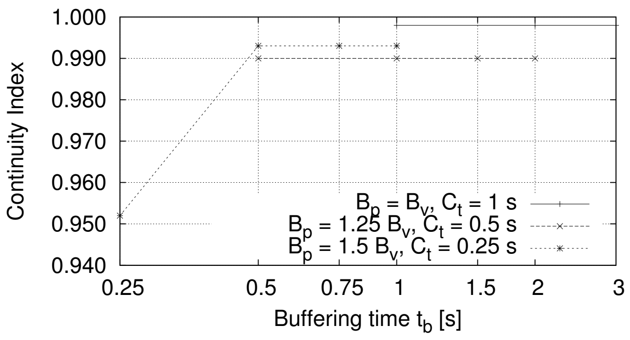

Fig. 6. Average video quality as a function of buffering time and the peer nodes bandwidth (videoconferencing scenario.)

communications can be achieved only at the price of high bandwidth requirements. Such results prompts further research towards more efficient encoding schemes tailored specifically for low delay applications such as videoconferencing.

Acknowledgments

Part of this work was supported by project ARACHNE, a PRIN funded by the Italian Ministry of Education and Research.

5. REFERENCES

[1] C. Fragouli, J. Le Boudec, and J. Widmer, "Network coding: an instant primer," *Computer Communication Review*, vol. 36, no. 1, pp. 63, 2006.

[2] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1655–1666, 2007.

[3] M. Toldo and E. Magli, "A resilient and low-delay p2p streaming system based on network coding with random multicast trees," in *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop on*. IEEE, 2010, pp. 400–405.

[4] A. Fiandrotti, V. Bioglio, M. Grangetto, E. Magli, and R. Gaeta, "Band codes: Complexity adaptive network coding for p2p video streaming," in *Multimedia and Expo (ICME), 2012 IEEE International Conference on*. IEEE, July 2012.

[5] P.A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*. The University; 1998, 2003, vol. 41, pp. 40–49.

[6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.