Author manuscript, published in "LCN 2010, the 35th IEEE Conference on Local Computer Networks (LCN) (2010)"

inria-00506065, version 1 - 27 Jul 2010

Performance Analysis of a High-Performance Real-Time Application with Several AL-FEC Schemes

Kazuhisa Matsuzono*, Jonathan Detchart†, Mathieu Cunche†, Vincent Roca† and Hitoshi Asaeda*
*Graduate School of Media and Governance,
Keio University, 252-8520 Kanagawa, Japan
Email: kazuhisa@sfc.wide.ad.jp, asaeda@wide.ad.jp
†INRIA, Planète research team, France
Email: {mathieu.cunche, jonathan.detchart, vincent.roca}@inria.fr

***Abstract*—Real-time streaming applications typically require minimizing packet loss and transmission delay so as to keep the best possible playback quality. From this point of view, IP datagram losses (e.g. caused by a congested router, or caused by a short term fading problem with wireless transmissions) have major negative impacts. Although Application Layer Forward Error Correction (AL-FEC) is a useful technique for protecting against packet loss, the playback quality is largely sensitive to the AL-FEC code/codec features and the way they are used. In this work, we consider three FEC schemes for the erasure channel: 2D parity check codes, Reed-Solomon over GF(2^8) codes, and LDPC-Staircase codes, all of them being currently standardized within IETF. We have integrated these FEC schemes in the FECFRAME framework, a framework that is also being standardized at IETF, and whose goal is to integrate AL-FEC schemes in real-time protocol stacks in a simple and flexible way. Then we modified the Digital Video Transport System (DVTS) high-performance real-time video streaming application so that it can benefit from FECFRAME in order to recover from transmission impairments. We then carried out several performance evaluations in order to identify, for a given loss rate, the optimal configuration in which DVTS performs the best.**

I. Introduction

Real-time streaming applications are gaining increasing popularity. The deployment of a high-speed fiber in the "last mile" has brought high-quality streaming applications within reach of a large number of people. Recently, the development of high-quality streaming applications such as the Digital Video Transport System (DVTS) has attracted considerable attention [12], [13], [16]. Since DVTS can be used with a wide range of video cameras and ordinary PCs, it is in widespread use, for example, in e-learning, at international symposium, and in telemedicine [1].

Since high quality and high performance streaming applications consume a large amount of network bandwidth for data transmission – for instance, a single DVTS stream requires about 30 Mbps – they may compete with other data flows, thereby causing traffic congestion. When used over wireless networks, short term fading phenomena can cause large signal to noise ratio variations, which may prevent transmission errors from being recovered by lower layer FEC techniques. In both cases, such applications are susceptible to consecutive packet losses (or equivalently, erasures) and delay, both of which can lead to significant video quality degradations. These impairments can be critical, especially for mission-critical applications like telemedicine.

Packet loss resilience can be achieved in two ways: thanks to retransmissions or thanks to the addition of redundancy. The Automatic Repeat reQuest (ARQ) scheme, although widely-used in many transmission protocols (e.g. TCP), needs at least one Round Trip Time (RTT) to recover from a loss. This is an issue if this RTT exceeds the maximum acceptable delay of a real-time stream. Another problem is the fact that ARQ does not scale well in case of multicast or broadcast diffusion. On the opposite, Application Layer Forward Error Correction (AL-FEC) codes rely on error correcting codes for the packet erasure channel. With this type of channel, packets are either received without any error or totally lost during the transmission. The use of AL-FEC codes prevents retransmission delays by preventively adding redundancy packets to the streaming data flow. Thanks to this redundancy, a certain number of missing source packets can be recovered at the receiver side without the need to ask the sender for lost packets. Therefore AL-FEC is commonly used for real-time streaming applications like a video conference systems.

More precisely, an AL-FEC encoder adds $n - k$ repair packets to each block of k source packets, which allows a receiver to rebuild all of the k source packets if more than k packets are received among the n packets sent. Maximum-Distance Separable (MDS) codes, such as Reed-Solomon codes [6], can recover all missing packets from any set of exactly k packets, while LDPC (Low Density Parity Check) codes [8] need to receive a small number of extra packets in addition to k. The performance of AL-FEC codes and the associated codecs (their software implementation), in terms of recovery capability and encoding/decoding delay largely varies, depending on the code intrinsic properties (e.g. is it an MDS code or not?), on the details of the codec (e.g. which kind of decoding algorithm is used, in particular with LDPC codes),

This work was supported by the National Institute of Information and Communications Technology (NICT), Japan, under Dynamic Network Project. This work was supported by the French ANR grants 2006 TCOM 019 (CAPRI-FEC project) and ANR-09-VERS-019-02 (ARSSO project).

inria-00506065, version 1 - 27 Jul 2010

the way transmissions occur (e.g. are packets sent sequentially or in a random order?), and the code parameters (e.g. the encoding block length n and the code rate k/n). Since real-time applications are sensitive to both delay and packet losses, and since these two aspect are contradictory (the larger the block size, the better the protection, but the higher the delay), it is usually necessary to find a trade-off. In the past, many studies have been conducted to improve the performance in terms of either codec algorithm, or packet scheduling, or parameter adjustment. However, they usually did not compare actual AL-FEC techniques with an operational, high throughput real-time application. A low bit-rate application is more commonly used, which cannot highlight some phenomena like CPU load.

In this paper, we analyze the suitability of three AL-FEC codes and codecs (i.e. their software implementation) for high-performance real-time applications. More precisely, we considered the 2D parity check codes, the Reed-Solomon over GF(2^8) codes, and LDPC-staircase codes, all of them being standardized within the `fecframe` and `rmt` IETF working groups. We have also implemented the FECFRAME framework, following the current IETF specification [3] of the `fecframe` IETF working group. The goal of this framework is to integrate FEC schemes to real-time protocol stacks and applications in a simple and flexible way. Then we modified the Digital Video Transport System (DVTS) high-performance real-time video streaming application so that it can benefit from the FECFRAME framework. Finally, we carried out several experiments in order to identify, for a given loss rate, the configuration in which DVTS performs the best.

The remainder of this paper is organized as follows: Section II introduces related work and AL-FEC codes. Section III describes the overview of DVTS with FEC scheme. Section IV evaluates the performance of DVTS with FEC scheme, and we conclude in Section V.

II. Application Layer FEC for Real-Time Streaming

A. Related Work

Besides the diversity in the underlying FEC encoding/decoding techniques, different FEC mechanisms are proposed and analyzed to improve the performances. For instance, the impacts of packet scheduling and loss distribution on FEC performances are studied [18]. In [19], four different FEC techniques, that utilize `XOR` (eXclusive-OR) and RSE erasure codes, are adopted according to the network conditions. In [11] the performances of the LDPC-staircase codes are studied when using a hybrid IT/ML decoding scheme, both from the erasure recovery capability and decoding complexity viewpoints. This work is one of the foundations of the current paper, with the difference that in our work we are considering real-time flows, with different goals.

Concerning the parameter adjustment techniques (e.g. code rate), many studies have been conducted so far [14], [17], [15]. However, in the context of high-performance real-time applications, practical results and evaluations as a function of the FEC codes have not been fully studied. Many past studies rely on streaming software featuring a low bit-rate (i.e. less than 10 Mbps) and non-real-time flows (i.e. non strictly enforced). In that case phenomena like the impact of the codec in terms of CPU load are not visible.

B. AL-FEC Codes

Let us focus now on the three AL-FEC codes being considered in our work:

1) 2D Parity Check Codes: The 2D parity check codes (2D codes) are classified as "simple codes" [4], which suggests that these codes are effective for recovering packet losses under very low loss conditions. For example, a simple way of adding protection against a single packet loss consists in creating a parity (`XOR`) of a certain number of source symbols (where a symbol is a fixed size unit of data from the AL-FEC encoder/decoder point of view). The 2D codes are only slightly more complex than that: the k source symbols are arranged in a $p \times p$ square matrix, where $p = \sqrt{k}$ (we assume that p is an integral value). Then, for each row (resp. each column) a single repair symbol is created by computing the `XOR` sum for all source symbols in the row (resp. column). We can rebuild any row (resp. column) of the matrix from any p out of $p + 1$ symbols in the row (resp. column). Of course these codes have limits: they are not MDS codes (there are situations where k received symbols do not enable decoding to succeed), and they create constraints on both k ($p = \sqrt{k}$ must be an integral value) and on the number of repair symbols, $2p$ (said differently the code rate is necessarily equal to $\frac{k}{k+2p}$).

2) RSE Codes: The Reed-Solomon codes for the Erasure channel (RSE) [6] are classified as "small block codes" [4] (i.e. k, n are limited) and they are one of the most popular FEC codes. RSE codes are intrinsically limited by the Galois Field it uses (e.g. with GF(2^8), $n \leq 2^q - 1 = 255$). Since the cost of the finite field operations quickly increases with the finite field size [6], most of the practical applications restrain themselves to GF(2^8), even if it also limits n and k parameters. Yet, RSE codes are MDS codes which is a major asset (lost source packets can be recovered as soon as *exactly* k packets out of n are received for this block). In this paper we will focus on RSE codes over GF(2^8) only.

3) LDPC-staircase Codes: The LDPC-staircase codes [7], [8] are classified as "large block codes" [4] (i.e. k, n can be very large) and they are a particular case of regular Repeat Accumulate codes. Although they are not MDS codes, they have many advantages like the possibility to operate efficiently with large source k and n parameters, with a linear time encoding complexity. The parity check matrix H of LDPC-staircase codes is a $(n - k) \times n$ binary matrix which can be divided into two parts: the left side of the matrix, H_1, is composed of $n - k$ rows for k columns (i.e. the source symbols), while the right side of the matrix, H_2, is composed of $n-k$ rows for $n-k$ columns (i.e. the parity symbols). Here the H_2 matrix has a "staircase" (double diagonal) structure. It means that each parity symbol is the `XOR` sum of the previous parity symbol plus a very small number of source symbols. The H_1 matrix is filled in a fully regular way as follows:

inria-00506065, version 1 - 27 Jul 2010

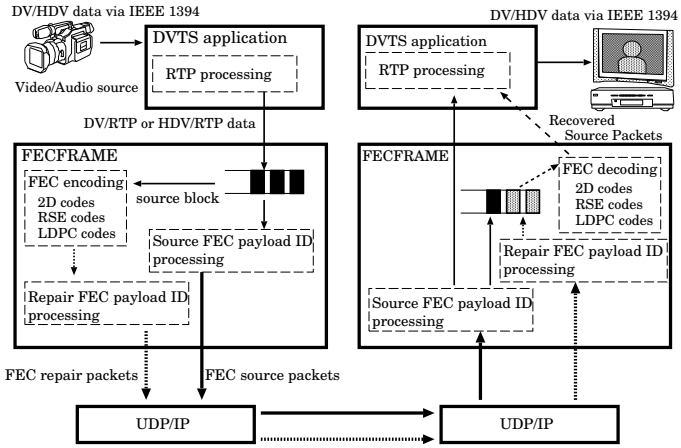

Fig. 1. The DVTS/FECFRAME architecture

1) Insert N_1 "1-s" randomly but evenly into each column;
2) Check there are at least two "1-s" per row. If not, add one or two extra "1-s" randomly to these rows.

In [10], [11], a hybrid ITerative/Maximum Likelihood (IT/ML) decoding was proposed to improve the erasure recovery capabilities (now close to that of MDS codes) while keeping a moderate decoding complexity. This algorithm combines the optimal correction capabilities of the ML decoding with the low complexity of the IT [9] decoding. It works as follows: start decoding with the linear complexity IT algorithm. If IT decoding fails and if it is known that no additional symbol will be received, switch to the more complex ML decoding (basically a Gaussian Elimination) and try to finish decoding. It is clear that a receiver can decide if and when ML decoding is used, depending on local criteria (e.g. battery or CPU capabilities), independently from other receivers, which is a great asset. In order to have good erasure recovery performances under IT/ML decoding, even with small n values, we adjust the N_1 parameter as follows [11]: if $n \leq 2^8$, $N_1 = 7$, otherwise $N_1 = 5$.

III. DVTS/FECFRAME Architecture

We now describe the DVTS/FECFRAME architecture (Fig. 1).

A. DVTS

The DVTS application can send and receive DV (Digital Video)/HDV (High-Definition Video) data over RTP [13]. The DV camera generates the DV/HDV stream, sends it over the IEEE 1394 interface to the sending host that runs the DVTS sending application. This latter encapsulates the DV/HDV frames into RTP packets and transmits them over the IP network. At the remote host, DVTS application upon receiving a DV/RTP or HDV/RTP packet, attaches an IEEE 1394 header to the received or reconstructed DV/HDV packets and transfers them to the DV/HDV recorder deck via the IEEE 1394 interface. A full DV stream consumes over 30 Mbps with standard NTSC quality video (525 lines and 29.97 fps). In case of HDV format (i.e. MPEG-2 TS), the full stream consumes about 25 Mbps.

B. FECFRAME Framework and FEC Schemes

The FECFRAME framework [3] defines a generic way of plugging FEC schemes (i.e. the codec implementing an AL-FEC code) dynamically into a real-time (or non real-time) protocol stack, for instance between the RTP and UDP layers. Thanks to this framework, a sending streaming application can generate and send FEC repair packets (i.e. redundancy), and a receiving streaming application can recover from lost source packets and pass them to the application as if they had been received normally. Several parameters need to be adjusted and communicated to both the sender and the receiver: 1) the nature of the FEC scheme (we consider 2D codes, Reed-Solomon and LDPC-staircase in this work), 2) FEC scheme specific parameters (e.g. with Reed-Solomon the finite field size, $m = 8$), 3) the symbol length (E), which is either fixed for the whole duration or adjusted per block, depending on the FEC scheme, 4) the source block length (k), 5) and the code rate k/n, or equivalently the total number of encoding symbols (source plus repair), n (subject to some constraints depending on the FEC scheme).

The encoding procedure, at the sender side, is as follows:

1) **Source block creation:** The FECFRAME instance receives (H)DV/RTP packets (known as Application Data Units, or ADU) from the application and stores them in a source block buffer for future encoding. To each ADU three bytes are prepended to form the source symbol (the source symbol is the unit of data used during the encoding/decoding process): 1 byte for the "flow ID" and 2 bytes for the ADU length. Then, since all source symbols need to be E bytes long, zero padding is added at the end of each ADU when needed. The Flow ID contains an integer that identifies the source flow (or application) to which this ADU belongs. This is necessary when several flows (several applications) are protected together by the same FECFRAME instance. The length contains an integer that indicates the ADU length, in bytes, without the three bytes and padding. This is necessary during decoding to strip padding.
2) **FEC encoding:** When the number of symbols stored in a source block buffer reaches k, or when the real-time constraints prevents the sender from waiting for additional ADUs (it can happen e.g. with variable bit rate flows), FEC encoding takes place. A certain number of repair symbols are created for this block, in accordance with the code rate (equivalently n) limitation.
3) **"Source" and "Repair FEC Payload ID" addition:** Then a "source FEC payload ID" (case of an ADU) is added at the end of the (H)DV/RTP packet[1] and transmission takes place. Similarly a "repair FEC payload ID"(case of a repair symbol) is prepended (rather than

[1]A mode exists where the (H)DV/RTP packet is sent without any modification (no "source FEC payload ID") for backward compatibility purposes. This mode has limitations and will not be considered in our work.

inria-00506065, version 1 - 27 Jul 2010

appended) to each repair symbol, and transmission takes place. The goal of these FEC payload ID are, for a receiver, to identify which block the symbol belongs to as well as the symbol "position" inside this block. The receiver can therefore gather all the symbols belonging to the same block and launch decoding if needed.

At the receiver side, the FECFRAME instance receives both FEC source and repair packets from the network. The decoding procedure is as follows:

1) **FEC source and repair packet storage:** Upon receiving a FEC source packet, the FECFRAME instance 1) passes a copy of it, without the source FEC payload ID, to the application immediately. It also 2) keeps a copy of it with other packets that belong to this block. Since the RTP data is immediately passed to the application, the application benefits from it without any additional waiting time. Upon receiving a FEC repair packet, the FECFRAME instance stores this packet in the buffer associated to the block it belongs to.
2) **FEC decoding:** If some source packets are missing in a source block and if the number of repair symbols received for that block is sufficient, FEC decoding is launched. Depending on the FEC code being used, it works as follows:
   - *2D codes:* When a packet belonging to the next source block arrives[2], a decision is taken: if at least k symbols are available for the current block, decoding is launched.
   - *RSE codes:* FEC decoding is launched as soon as exactly k source and repair symbols are received.
   - *LDPC-staircase codes:* LDPC-staircase codes require slightly more than k symbols for decoding to succeed. Based on experimental results [11], we set the threshold at $k \times (1.05)$, which corresponds to a very high decoding success probability and a low decoding complexity (IT decoding will significantly simplify the linear system to be solve during ML decoding). As soon as at least $k \times (1.05)$ symbols are received, FEC decoding is launched. Otherwise, when a packet belonging to the next source block arrives, a decision is taken: if at least k symbols are available, decoding is launched.

IV. Experimental Results

A. Experimental Setup and Performance Metrics

To evaluate the performance of DVTS with FEC scheme, we set up an experimental environment. The sender and the receiver are desktop machines under 32bit Linux operating system, running a modified DVTS that includes the FECFRAME framework and several FEC schemes. The sender uses a Pentium 4/2.4 GHz CPU/512 MB RAM and runs a Linux 32bit OS. The receiver uses a Pentium Core 2/1.66 GHz CPU/1 GB RAM and runs a Linux 32bit OS. The DVTS application is version hdvts1.0a [20]. The 2D codes, Reed-Solomon codes, and LDPC-staircase codes are those available in version 1.0.0 of the `OpenFEC.org` library [21].

The sending strategy consists in sending all FEC source packets first, then FEC repair packets. Additionally, in case of LDPC-staircase codes, the repair packets are sent in a random order, whereas for the two other codes, they are sent sequentially. During tests the uniform packet loss probability used is progressively increased from 0% to 51%, by 3% each time. The packets erased are randomly chosen, in accordance with this loss probability. During each test (that last 60 seconds each), the receiver then measures the recovery capabilities, the frame delays and the CPU resource usage. The CPU resource usage is also measured at the sender side.

Table I shows the parameters for each FEC code considered. The code rate is fixed and set equal to $2/3$ for all AL-FEC codes. $k = 170$ is chosen since it corresponds to $n = 255$, i.e. a value compatible with RSE over GF(2^8). Since we use DV format and the maximum DV/RTP packet size is 1372 bytes, we set $E = 1372 + 3$ bytes. This has to be compared to the maximum packet loss probability set deliberately to 51%. We can expect that as the packet loss probability approaches 33.3%, performance will degrade, and above this value and until 51%, performance will be rather poor, a full decoding of a source block being usually impossible (but partial decoding may happen, especially with the "small" 2D codes).

TABLE I
FEC PARAMETERS

FEC Codes	Symbol Length (bytes)	Code Rate	Source Block Length (k)	N_1
2D Codes	1375	2/3	16	none
RSE Codes	1375	2/3	170	none
LDPC Codes	1375	2/3	(1) 170	7
			(2) 500	5
			(3) 1000	5

B. Recovery Capabilities

The packet loss recovery capabilities can be evaluated by measuring the actual data loss percentage within the application (i.e. the DV/RTP packet loss percentage after loss correction within FECFRAME) as a function of the packet loss probability. Fig. 2 and Fig. 3 show that there remain unrecovered loss with a 30% packet loss probability (or higher). With 2D codes the situation is worse since about 0.03% unrecovered data losses happen even with a packet loss probability lower than 10%. Above 10%, unrecovered loss quickly increases. With RSE and LDPC codes, the unrecovered loss is null below approximately a 30% of packet loss probability. We see that RSE codes ($k = 170$) perform only marginally better than LDPC ($k = 170$) codes. Increasing LDPC's k parameter to $k = 500, 1000$ (we know that LDPC codes perform asymptotically well) improves marginally the erasure recovery capabilities (while increases the decoding delay which is clearly an issue).

[2]If we ignore possible packet reordering, this is the sign that no additional packet will be received for the current block. With possible packet reordering, the receiver should wait some more time.

inria-00506065, version 1 - 27 Jul 2010

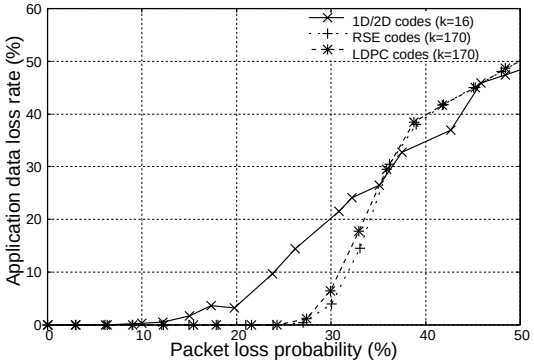

Fig. 2. Average data loss rate with 2D, RSE, and LDPC ($k = 170$) codes

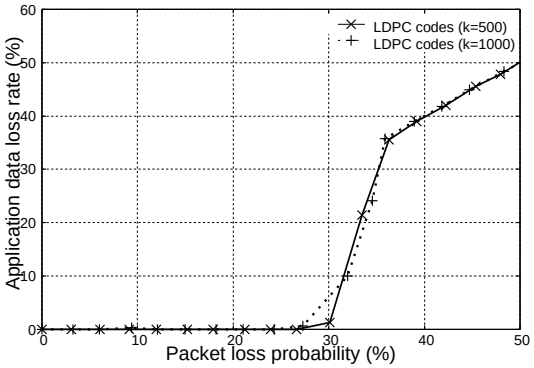

Fig. 3. Average data loss rate with LDPC codes ($k = 500, 1000$)

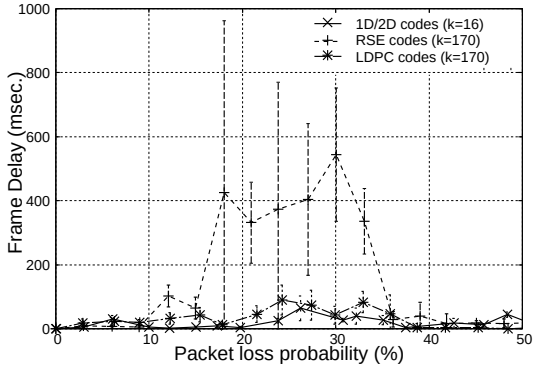

Fig. 4. Frame delay with 2D, RSE, and LDPC ($k = 170$) codes

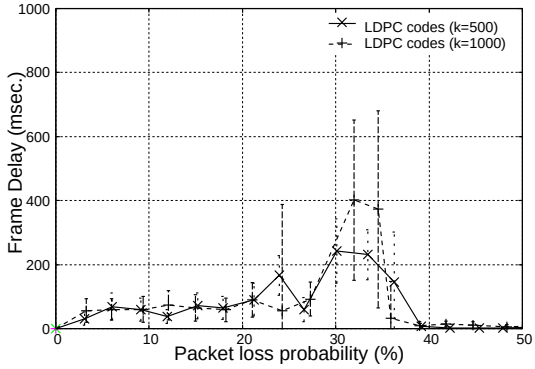

Fig. 5. Frame delay with LDPC codes ($k = 500, 1000$)

C. Frame Delay

We then evaluate the frame delay, during receiver's RTP processing, by measuring how long it take to the FECFRAME instance to get each DV/RTP packet, compared to a normal DVTS processing. To that purpose, we measure the number of delayed frames and their delay. The average frame delay in each test is then obtained by dividing the total delay time by the number of delayed frames. This average frame delay, plus the standard deviation above and below, as a function of the packet loss probability is shown in Fig. 4 and Fig. 5. With 2D codes, throughout the tests (from 0% to 51% loss probabilities), the maximum average frame delay turned out to be 63.7 msec (around a 26.2% packet loss probability). Otherwise, the average frame delay was usually lower than 50 msec.

With RSE codes, the average frame delay around a 12.1% packet loss probability was over 100 msec. Then around 30.1% packet loss probability, the average frame delay became 543.9 msec which is rather bad.

With LDPC codes ($k = 170$), when the packet loss probability was lower than 24%, the average frame delay was below 50 msec. Above 24% packet loss probability, the maximum average frame delay was about 89.4 msec. With larger LDPC codes ($k = 500, 1000$), the average frame delay was always greater than that of LDPC codes ($k = 170$). In particular, in high packet loss probability conditions (more than 24%), the average frame delay of LDPC codes with the large source block length was significantly greater. For instance, around a 30% packet loss probability, this average delay was about 241.6 msec ($k = 500$) and 401.5 msec ($= 1000$), respectively.

From these results we can say that in our use-case, small LDPC codes ($k = 170$) are a good solution since they offer good erasure capabilities (close to that of Reed-Solomon codes of similar dimension) while keeping the frame delay low, close to that of 2D codes.

D. Processing Load

In order to understand resource usage at the receiver side during FECFRAME processing, we measured the CPU load at one second interval. Fig. 6 and Fig. 7 show the average CPU load at the receiver, plus the standard deviation above and below, as a function of the packet loss probability. As seen from Fig. 6, the average CPU load of RSE codes quickly increases until the packet loss probability reaches approximately 30%. For instance, at 27.1% packet loss probability, the average CPU load at the receiver is 31.0%. For higher packet loss probabilities, this load decreases since the receiver has fewer and fewer opportunities to launch RSE decoding. On the opposite, with the 2D and LDPC-staircase codes, no matter the packet loss probability over the network, the average CPU load always remains below 8%.

inria-00506065, version 1 - 27 Jul 2010

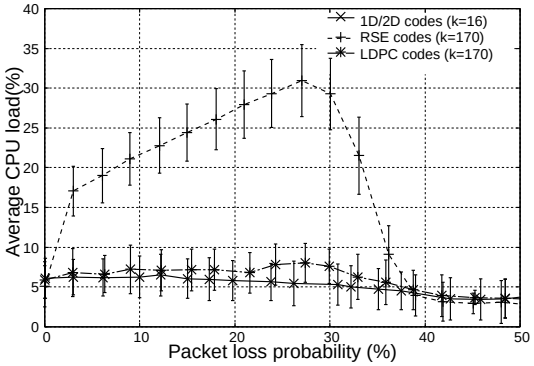

Fig. 6. CPU load at the receiver side, with 2D, RSE, and LDPC ($k = 170$) codes

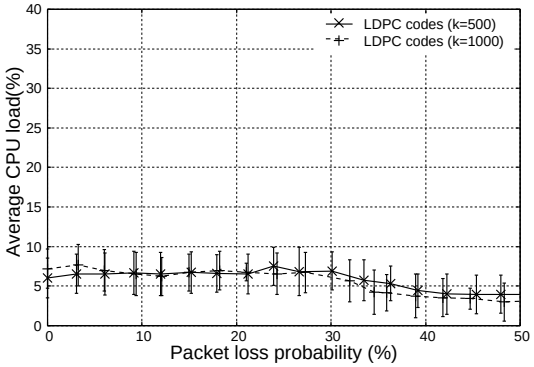

Fig. 7. CPU load at the receiver side, with LDPC codes ($k = 500, 1000$)

We also measured the CPU load at the sender side during FECFRAME processing. Table II shows the average CPU load and the corresponding standard deviation. We see that RSE encoding create a significant CPU load, compared to 2D and LDPC-staircase codes.

E. Discussions and Recommendations

Based on the experimental results in terms of recovery capabilities, frame delay, and processing load, we can identify the optimal configuration that achieves the best performance. In presence of a certain packet loss probability, it is necessary to find a well-balanced point between the recovery capabilities and frame delay. In general, in order to absorb the arrival delay of the source packet, a receiver requires a packet buffer of a certain length at the cost of the real-time performance. If the packet arrival delay exceeds the acceptable range determined by the buffer length, the packet is discarded. Therefore, an appropriate tuning of this buffer length is vital.

TABLE II
CPU LOAD AT THE SENDER SIDE

	2D codes	RSE codes	LDPC codes		
k, code rate	16, 2/3	170, 2/3	170, 2/3	500, 2/3	1000, 2/3
ave. CPU load (%)	14.1	74.3	12.0	11.0	12.6
std deviation	6.0	9.0	6.3	6.7	6.2

Within less than 10% of packet loss probability, RSE and LDPC-staircase codes recover almost all the lost source packets. The 2D codes continually causes about 0.03% data loss rate because of the small source block length ($k = 16$). In the case of 2D, RSE, and LDPC ($k = 170$) codes, the receiver needs the equivalent buffer length of 100 msec in order to absorb the generated frame delay. However Fig. 8 shows that LDPC codes with larger source block lengths ($k = 500, 1000$) causes some frames to be discarded with the equivalent 100 msec buffer. With an equivalent 200 msec buffer (not shown), there is no discarded frame any more. In this context we can conclude that RSE and LDPC-staircase ($k = 170$) codes are suitable in the presence of low and non-burst packet loss conditions, in terms of both video quality and real-time performance. With a low spec machine, LDPC-staircase ($k = 170$) and 2D codes should be preferred because of the modest processing load generated.

In presence of more than 10% packet loss probability, 2D codes cannot recover lost source packets, while RSE and LDPC-staircase codes fully recover them up to around 30% packet loss probability. As seen from Fig. 9, when using an equivalent 100 msec buffer, the number of discarded frames in all cases increases as the packet loss probability increases. This is particularly true for RSE codes that behave badly. On the opposite, LDPC-staircase ($k = 170$) codes are the codes that perform the best (not considering 2D codes). In Fig. 10, with an equivalent 200 msec buffer, the number of discarded frames considerably decreases except for RSE codes that still show poor performance. To a lesser extent, with high packet loss probabilities, LDPC codes ($k = 500, 1000$) feature a large number of discarded frames because of their important decoding time (more packets are needed before decoding can start). However, if a receiver can afford a large equivalent buffer length, at the cost of real-time performance, LDPC-staircase codes with large block lengths can achieve the higher recovery capability in high and bursty packet loss conditions which can be a significant asset.

To conclude, LDPC-staircase ($k = 170$) codes are good choices in terms of recovery capabilities, real-time performance and processing load, no matter the incoming packet loss rate.

V. CONCLUSIONS AND FUTURE WORKS

This work analyses the benefits of adding three AL-FEC codes, namely the 2D parity check, Reed-Solomon (RSE) over GF(2^8) and LDPC-staircase codes, in a high-performance, high throughput real-time video streaming application, the Digital Video Transport System (DVTS). More precisely DVTS has been modified, following the FECFRAME approach [3] to integrate AL-FEC codes. Based on the experimental results in terms of recovery capabilities, frame delay and processing load, we have identified the optimal configuration for a given incoming loss rate. More precisely, under low packet loss conditions, all FEC codes achieve high quality video playback since erasures are recovered within their real-time constraints. However, from the processing load point of

inria-00506065, version 1 - 27 Jul 2010

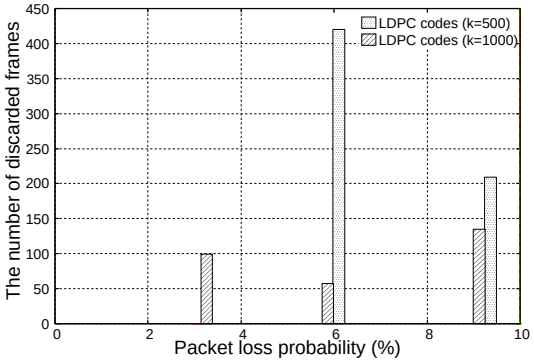

Fig. 8. The number of discarded frames when using the equivalent buffer of 100 msec, in the presence of the packet loss probability from 0% to 10%

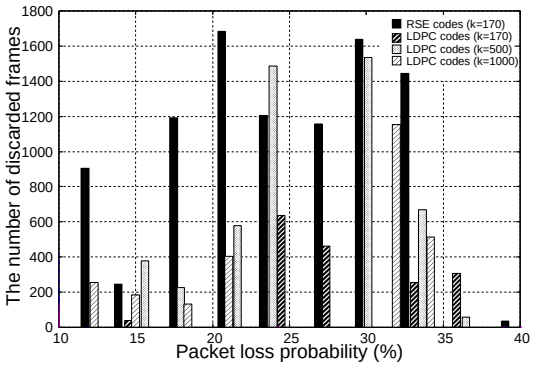

Fig. 9. The number of discarded frames when using the equivalent buffer of 100 msec, in the presence of the packet loss probability from 10% to 40%

view, RSE codes create an important CPU load compared to the other codes. Under high packet loss conditions, the 2D and RSE codes no longer perform well in terms of the recovery capability and real-time performance. On the opposite, the LDPC-Staircase ($k = 170$) codes enable DVTS to achieve almost optimal performance.

In future works, we plan to evaluate the FEC schemes in an heterogeneous communication environment. We will also develop FEC scheme to dynamically adjust both FEC parameters (i.e. source block length and code rate) and the streaming bit-rate based on the network conditions in order to further improve performance.

REFERENCES

[1] N. Nakashima, K. Okamura, JS. Hahm, YW. Kim, H. Mizushima, H. Tatsumi, BI. Moon, HS. Han, YJ. Park, JH. Lee, SK. Youm, CH. Kang, and S. Shimizu, "Telemedicine with digital video transport system in Asia-Pacific area," Proc. of the 19th International Conference on Advanced Information Networking and Applications, March 2005.

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

[3] M. Watson, "Forward Error Correction (FEC) Framework," draft-ietf-fecframe-framework-09.txt (work in progress), IETF Internet draft, July 2010.

[4] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast," IETF RFC 3453, December 2002.

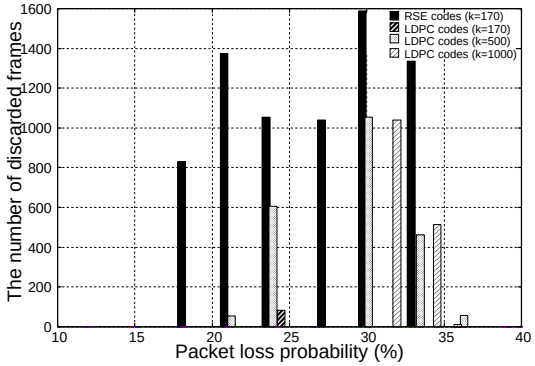

Fig. 10. The number of discarded frames when using the equivalent buffer of 200 msec, in the presence of the packet loss probability from 10% to 40%

[5] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," ACM Comput. Commun, Vol.27, no.2, pp.24-36, April 1997.

[6] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes" IETF RFC 5510, April 2009.

[7] D. MacKay, "Information Theory, Inference and Learning Algorithms," Cambride University Press, ISBN : 0521642981, 2003.

[8] V. Roca, C. Neumann and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes," IETF RFC 5170, June 2008.

[9] V. Zyablov and M. Pinsker, "Decoding complexity of low-density codes for transmission in a channel with erasures," Translated from Problemy Peredachi Informatsii, 10(1), January 1974.

[10] M. Cunche and V. Roca, "Improving the ecoding of LDPC codes for the packet erasure channel with a hybrid Zyablov iterative decoding/Faussian elimination scheme," Research Report 6473, INRIA, March 2008.

[11] M. Cunche and V. Roca, "Optimizing the Error Recovery Capabilities of LDPC-staircase Codes Featuring a Gaussian Elimination Decoding Scheme," Proc. of IEEE International Workshop on Signal Processing for Space Communications (SPSC'2008).

[12] A. Ogawa, K. Kobayashi, K. Sugiura, O. Nakamura, and J. Murai, "Design and Implementation of DV based video over RTP," Proc. of Packet Video Workshop 2000, May 2000.

[13] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann. "RTP Payload Format for DV (IEC 61834) Video," IETF RFC 3189, January 2002.

[14] H. Wu, M. Claypool, and R. Kinicki, "Adjusting Forward Error Correction with Quality Scaling for Streaming MPEG," Proc. of ACM NOSSDAV '05, June 2005, Washington, USA.

[15] K. Matsuzono, K. Sugiura, and H. Asaeda, "Adaptive Rate Control with Dynamic FEC for Real-Time DV streaming," Proc. of IEEE Globecom'08, December 2008.

[16] C. Bao, X. Li, and J. Jiang, "Scalable Application-Specific Measurement Framework for High Performance Network Video," Proc. of ACM NOSSDAV '07, Urbana, Illinois USA.

[17] J.C. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive FEC-based error control for Internet telephony," Proc. of IEEE INFOCOM '99, New York, USA, March 1999.

[18] C. Neumann, V. Roca, A. Francillon, and D. Furodet, "Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances: Observations and Recommendations," Proc. of CoNEXT'05, Toulouse, France, October 2005.

[19] C. Lamoriniere, A. Nafaa, and L. Murphy "Dynamic Switching Between Adaptive FEC Protocols for Reliable Multi-Source Streaming," Proc. of IEEE Globecom'09 Hawai USA, November 2009.

[20] "DVTS: Digital Video Transport System, or DV Stream on IEEE1394 Encapsulated into IP", http://www.sfc.wide.ad.jp/DVTS/.

[21] "OpenFEC.org: because open, free AL-FEC codes and codecs matter", http://openfec.org.