Integrated Error Management for Media-on-Demand Services

Pål Halvorsen, Thomas Plagemann
UniK, University of Oslo
P.O. Box 70, N-2027 KJELLER, Norway
{paalh, plageman}@unik.no

Vera Goebel
Department of Informatics, University of Oslo
P.O. Box 1080, Blindern, N-0316 OSLO, Norway
goebel@ifi.uio.no

Abstract— **Data servers for multimedia applications like News-on-Demand represent a severe bottleneck, because a potentially (very) high number of users concurrently retrieve data with high data rates. In the Intermediate Storage Node Concept (INSTANCE) project, we develop a new architecture for Media-on-Demand servers that maximizes the number of concurrent clients a single server can support. Traditional bottlenecks, like copy operations, multiple copies of the same data element in main memory, and checksum calculation in communication protocols are avoided by applying three orthogonal techniques: zero-copy-one-copy memory architecture, network level framing, and integrated error management. In this paper, we describe design, implementation, and evaluation of our integrated error management mechanism. Our results show that the reuse of parity information from a RAID system as forward error correction information in the transport protocol reduces the server workload and enables smooth playout at the client.**

I. Introduction

In the last years, there has been a tremendous growth in the use of Internet services. In particular, the world-wide web and applications like News-on-Demand (NoD) have become very popular. Thus, the number of users, as well as the amount of data each user downloads from servers in the Internet, is rapidly increasing. The usage of multimedia data to represent information in a user-friendly way is one important reason for these two developments. Today, contemporary mid-price personal computers are capable of handling the load that such multimedia applications impose on the client system. However, the potentially (very) high number of concurrent users that download data from NoD, Video-on-Demand, or more generally Media-on-Demand (MoD), servers represents a generic problem for this kind of client-server applications. There are two basic orthogonal approaches for this problem: (1) development of an architecture for a single server that makes optimal use of a given set of resources, i.e., maximize the number of concurrent clients a single server can support; and (2) combination of multiple single servers, e.g., in a server farm or cluster, to scale up the number of concurrent users.

In the *Intermediate Storage Node Concept* (INSTANCE) project [31], we concentrate on the first approach developing a new architecture for single servers. This architecture is optimized to support concurrent users in MoD-like applications retrieving multimedia data from the server. Thus, the task of reading data from disk and transmitting it through the network to remote clients with minimal overhead is our challenge and aim. Our target application domain comprises MoD-like applications in the smaller scale, like MoD for entertainment in hotels or airplanes, and in the medium scale, like MoD provided by city-wide cable companies or pay-per-view companies. Whether the INSTANCE approach is also beneficial for large scale systems like NoD services from CNN or BBC is subject to further research.

It is a well known fact, that *operating systems are not getting faster as fast as hardware* [25], and that commodity operating systems represent today the major performance bottleneck in MoD servers. The crucial issues that contribute to this situation include copy operations, context switches, multiple copies of the same data element in main memory, and checksum calculation in communication protocols [32].

The key idea of INSTANCE is to avoid the above mentioned bottlenecks and improve the server performance by combining the following three orthogonal techniques in a radically new architecture [18]:

- *Zero-copy-one-copy memory architecture*: Memory copy operations have been identified as a bottleneck in high data rate systems. Several zero-copy architectures removing physical data copying have been designed to optimize resource usage and performance using shared memory, page remapping, or a combination of both [32]. These approaches reduce resource consumption of individual clients, but concurrent clients requesting the same data require each its own set of resources. Traditional broadcast or multicast is an easy way of dealing with per client resource allocation, but startup delay might be a problem. To minimize the startup delay, a couple of broadcast partitioning schemes are proposed [32]. Still, zero-copy and delay-minimized broadcasting only reduce the per-data or per-client resource usage, respectively. To optimize both, we integrate both mechanisms to have no physical in-memory copy operations and to have only one shared copy of a data element in memory.
- *Network level framing mechanism*: Each time a client retrieves data from a server, the data is processed through the communication system protocols executing the same operations on the same data element several times, i.e., for each client. To reduce this workload, we regard the server as an *intermediate node* in the network where only the lower layers of the protocol stack are processed. When new data is sent to the server to be stored on disk, only the lowest three protocol layers are handled and the resulting transport protocol packet is stored on disk. When data is requested by remote clients, the transport level packet is retrieved from disk, the destination port number and IP address are filled in, and the checksum is updated (only the new part for the checksum, i.e., the new addresses, is calculated). Thus, the end-to-end protocols which perform the most costly operations in the communication system, especially the transport level checksum, are almost completely eliminated.
- *Integrated error management scheme*: When transferring data from disk in a server through the network to remote clients, the correctness of the information is checked multiple times, i.e., the error management functionality is redundant (see Figure 1). In INSTANCE, we integrate the error management in a disk ar-

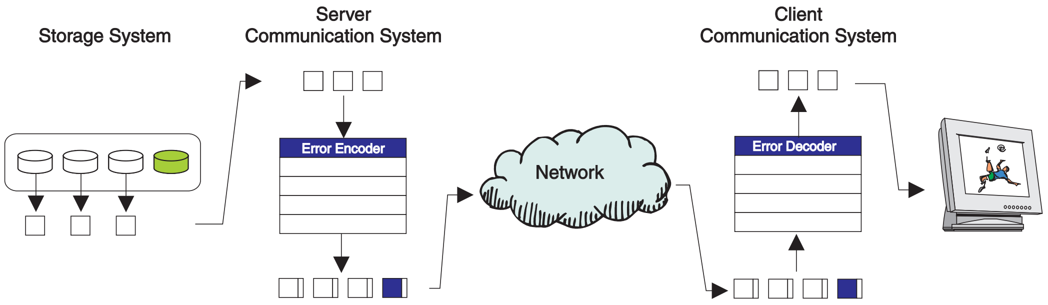


Fig. 1. Traditional error management.

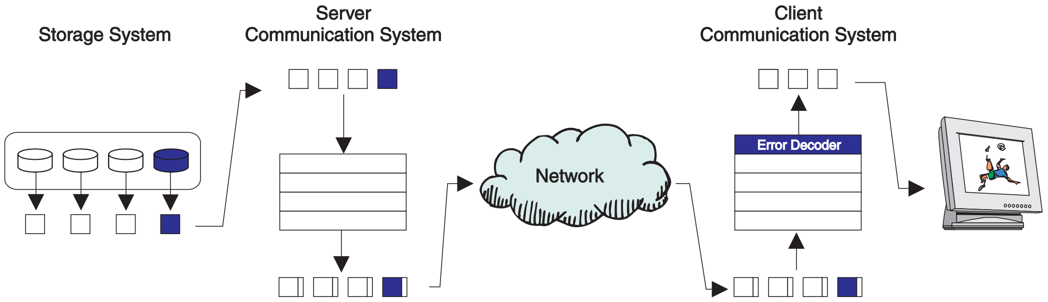


Fig. 2. Integrated error management.

ray and a forward error correction (FEC) scheme in the communication system. Opposed to the traditional data read from a redundant array of inexpensive disks (RAID) system where the parity information is only read when a disk error occurs, we retrieve also the redundant error recovery data, i.e., making a RAID level 0 read operation. All data is passed over to the communication system, where the parity information from the RAID system is re-used as FEC information over the original data (see Figure 2). Thus, by using the same parity information in both subsystems, the FEC encoder can be removed from the server communication system, and both memory and CPU resources are made available for other tasks.

This paper focuses on design, implementation, and evaluation of the integrated error management scheme, that enables us to reduce the server workload and support a faster data corruption recovery giving the clients a smooth multimedia presentation.

The rest of this paper is organized as follows: In Section II, we discuss the requirements that have to be fulfilled when successfully combining error management in storage and network system. Section III describes our (experimental) evaluation of different design alternatives. Our prototype design is described in Section IV, and results from a performance measurement of our prototype are presented in Section V. Finally, we summarize and conclude the paper in Section VI.

II. Background and Requirements

To design an integrated error management mechanism, the recovery scheme must be able to correct both failures occurring in the storage system and data corruption and losses during a transmission through the network. In the following subsections, we describe the different system error models, and we use these models to present our requirements.

A. Storage System Error Model

The primary medium for long-term on-line storage of data is the magnetic disk, because it usually is very reliable. Nevertheless, although disk failures are much less frequent than other system errors, the disk devices themselves may sometimes fail. For example, the high-end SCSI disk drives from Seagate (Barracuda and Cheetah) [36] have a mean time between failure of at least 10^6 hours. Furthermore, the disks have one seek error for every $10^7 - 10^8$ bits read, a recoverable error rate of 1 in $10^{10} - 10^{11}$ bits read, and an unrecoverable error rate of 1 in $10^{14} - 10^{15}$ bits read.

A single disk usually does not provide the requested data rate retrieving multimedia data for several different streams each requiring a high bandwidth [19]. Hence, performance can be increased through disk parallelism using techniques like data striping, and disks combined in an array are often used for delay sensitive applications on high performance storage servers. However, as the number of disks increases, the mean time to failure is also reduced. A variety of disk organizations have been proposed where the different levels of RAID [14], [26] are often used to increase reliability and performance.

B. Network Error Model

Transmitting a data element across a network can introduce errors in the bit stream, and each transmission medium has its own typical error source [17]. Any signal carried on a transmission medium is affected by *attenuation*, *noise*, *bandwidth*, and *distortion* where the amount of bit errors depends on several different parameters like the underlying network technology and the length of the network. Wired networks are today usually very reliable. For example, the 10BASE-T (Ethernet) allows one corrupted bit per 10^8 bits transmitted, but typical performance is 1 in 10^{12} or better [1]. 100BASE-T and 1000BASE-T both perform with less than one erroneous bit in 10^{10} [16]. Wireless networks are more exposed to noise than wired networks, and bit errors are therefore more common. Some frequencies are more susceptible to atmospheric disturbances such as rain attenuation, and mobile receivers may be more vulnerable to distortion and shadowing, e.g., blockage by buildings. Typical bit error rates in satellite networks are in the order of 1 error per 10^7 bits transmitted or less [2].

Besides bit errors, whole packets may be lost, e.g., due to *congestion*. This is the predominant cause of loss [30]. Unfortunately, there exists no precise knowledge about error occurrence or error models in the Internet, and only a few measurements are reported in literature. In [10], 384 Kbps and 1 Mbps MPEG video is transmitted over the Internet with different packet sizes using UDP/IP. The average packet loss rates vary between 3% and 13.5% with greater loss rates during periods of high network activity, and on average, 1.7% to 15.4% of the packets arrived out-of-order. Measurements of packet loss transmitting 320 B audio packets between southern France and London is presented in [9]. Consecutive losses of up to 30 packets are reported both for unicast and multicast connections, but the vast majority loss is one or two consecutive packets. Furthermore, in a study described in [40], the packet loss in an MBone multicast network, transmitting small packets (320 B and 640 B) of audio data, is measured. On average, most losses occur at the receiver side. There are a significant amount of both small burst (consecutive) losses and single, isolated losses. Only a few long bursts, i.e., spanning over 100 packets, of congestion loss were observed.

The network error models indicate that different kinds of data corruption may occur in a client-server based system, and some receivers will therefore experience packet loss or destruction during data retrieval from a remote server [30]. Different recovery techniques have been developed to more effectively deal with different sources of errors [21]. For example, noise is of short duration and should have no lasting effect on error rate. Hence, error recovery can focus on a small time window. Congestion, on the other hand, may last for an extended time, requiring mechanisms to manage multiple packet losses over a longer time period.

C. Correction Scheme Requirements

In our design of a cost effective, high performance MoD server, the integrated recovery scheme must be able to correct the described data losses in both storage and communication system. The most obvious and used method of detecting and restoring corrupted data is to calculate some redundant data over a group of information frames. Each frame consists of one or more bits and is referred to as a *symbol*. The composite message of information symbols and corresponding parity symbols is referred to as a *codeword*. In general, we differentiate between two kinds of data corruption: *erasures*, where the location of the damaged data is known, and *errors*, in which the position of the corrupted data is unknown. Thus, we have an erasure if a disk fails or a packet containing a sequence number is lost, but a packet damaged by noise changing some bit values is an error. To summarize, our integrated mechanism for an MoD server must have the following properties:

1. *Error/erasure correcting capability:* The coding scheme must be able to correct both bursts of bit errors and large groups of erasures. In the network, bit errors occur for example due to noise (error), and packets are lost due to congestion (erasure). In the storage system, a disk may fail (erasure). Thus, the striping units in the RAID system and the network packets in the communication system are units of recovery, and the correcting capability of the error management scheme must therefore suit all kinds of recovery.
2. *Decoding throughput (performance):* The decoding[1] algorithm must be efficient and fast enough to be able to reconstruct the multimedia data at the client side in time for playout and still have available resources for other tasks. For example, an audio stream requires about 16 Kbps for telephony and 1.4 Mbps for CD-quality. A video stream's requirement ranges from approximately 1.2 Mbps for MPEG to 20 Mbps for compressed and more than 1 Gbps for uncompressed HDTV [23]. The current MPEG 2 Digital Versatile Disk (DVD) standard has an average bit rate of 3.5 Mbps, and the maximum bit rate is 9.8 Mbps [22].
3. *Amount of redundant data:* The correction information that is transmitted to the remote client together with the original data imposes higher throughput needs. Since the throughput requirement of multimedia streams is already very high and an enlarged amount of transmitted data further increases network utilization, which potentially increases congestion and noise-caused damage, we should find a scheme which minimizes the amount of redundant information. However, this is a contradictory requirement compared to property 1, because the correction capability depends on the amount of redundant parity information.
4. *Applicable within a single file:* A traditional RAID system performs recovery based on stripes regardless of which file the data belongs to. We want only to transmit the original information and the stored parity information of a single file to the client. Correction schemes which calculate parity information based on disk setup regardless of which files the disk blocks (or striping units) belong to are not suitable to serve our purpose.
5. *Quality-of-Service (QoS) determined decoding cost:* The use of FEC means that all receivers are forced to receive and decode the data stream consisting of application data and redundant data; including those that do not need the improved probability of an error free connection or better quality of the multimedia playout [15]. We require that the decoding cost should depend on the application QoS requested by the client and the QoS provided by the underlying network itself. In particular, the cost of the decoding function should be negligible in the following two cases: (1) the network provides already error free transmission, and (2) a receiver is willing to tolerate errors in the multimedia data (to reduce the load on other client resources).

III. Design Alternatives

In order to find a mechanism that meets the above mentioned requirements, we have evaluated existing techniques for the applicability in INSTANCE.

A. Existing Recovery Schemes

As we can see from the error models described in Sections II-A and II-B, different kinds of data destructions and losses occur in the different subsystems. Distinct mechanisms are therefore applied for recovery. There exist several schemes ranging from single parity calculations using XOR to more complex algorithms like Reed-Solomon (RS) or Convolutional codes [17], [35]. Following, we describe recovery mechanisms for storage and communication systems.

[1] Please note that encoding throughput is less important, because the encoding operation is only done off-line, and the parity information is stored on and retrieved from disk.

TABLE I
RAID LEVELS.

	Parity unit	Parity location	Recovery capability
Level 0	None	None	None
Level 1	Mirroring	Disk duplicating	One disk failure per parity disk
Level 2	Bit	Predefined disk	One disk failure
Level 3	Bit	Predefined disk	One disk failure
Level 4	Block	Predefined disk	One disk failure
Level 5	Block	Distributed	One disk failure
Level 6	Block	Distributed	Two disk failures

A.1 Storage System Recovery

To detect errors during an I/O operation, the disk controller attaches checksums to each sector derived from the data on the respective blocks. When performing a read operation, the checksum is computed again and compared to the stored value. If the two checksums are different, the controller rereads the requested data, and if the error continues to appear, the controller will signal a read failure. Thus, transient errors, e.g., caused by dust on the head, and seek errors can often be corrected by the controller, but permanent errors, e.g., physically damaged disk blocks, will cause a data loss if no other error recovery mechanism is used.

Both performance and reliability can be increased through disk parallelism in RAID systems, and the applied solution to restore data if a disk fails is to introduce redundant data for error recovery. As shown in Table I describing the RAID levels [14], [26], adding correcting information can be done in several ways. If the read operation fails, the disk controller first tries to re-read the disk block, but if the error is permanent, it is corrected with help of the error correction scheme in the particular RAID level. Since the controller knows which disk failed, the parity data and the remaining data from the other disks can be used to determine the bit values of the lost data.

In today's RAID solutions, XOR correction schemes are usually applied for reliability, but there are also other schemes proposed for disk arrays in general or as improvements of the RAID mechanisms. For example, the *EVENODD* [5] double disk failure scheme uses a two-dimensional XOR parity calculation which shows good results both regarding redundant information overhead and performance. In [19], the problem of designing erasure-correcting binary linear codes protecting against disk failures in large arrays is addressed. Several two-dimensional erasure-correcting solutions capable of correcting multiple disk failures are presented. RS RAID systems are described in [33] where m disk failures can be recovered by applying m redundant disks.

A.2 Error Management in Communication Systems

In the communication system, different kinds of data losses occur depending on the underlying network technology, the length of the network, etc., and irrespective of the kind of errors that occur, a mechanism is needed to obtain a copy of the correct information. The applied data correction mechanism must be able to correct the occurring damages, and at the same time fulfill all application requirements, e.g., throughput and latency. In general, there are two basic approaches to achieve this [4], [17]: *automatic repeat request (ARQ)* or *FEC*.

Usually, ARQ is more efficient, i.e., using a checksum for error detection followed by a retransmission, because the complexity of FEC algorithms makes the execution of these procedures too slow and expensive. Nevertheless, the attractiveness of FEC has changed during the past decades. Significant changes in communication technology for switched networks reducing the amount of bit errors makes congestion the dominant source of errors. This enable the use of more efficient codes [21]. Correction codes are therefore used for data transmission if retransmission is not available or not preferred as described in the following situations:

- Using ARQ as the end-to-end error control scheme in multicast performs poorly, because it does not scale well. The transmitter complexity is proportional to the amount of receivers and requires a lot of buffer space for potential retransmissions. FEC-based multicast protocols scale much better, since packet losses will be handled at the receiver and do not require retransmissions in the multicast tree. Thus, FEC is a good solution for broadband communications where data is multicast over wide area networks to a large group of clients [12], [15], [21], [24], [34].
- For real-time data services, or time sensitive and delay critical services in general, ARQ is not suitable [3], [12], [20], [23]. The feedback channel overhead and the retransmission latency are the most important limitations. For example, a retransmitted video frame may be too old for presentation when arriving at the client side. Low latency is necessary for applications supporting multimedia playouts or human interactions in which the delay and delay jitter caused by retransmission will be unacceptable. Consequently, FEC may be an appropriate error recovery scheme for time sensitive applications providing reliability without increasing end-to-end latency.

Several codes have been used for FEC. Convolutional code [17], [35] with Viterbi decoding has been a de facto standard for over 30 years, and it is a good scheme for correcting randomly occurring errors. It is used in many real-world systems [35], especially in satellites and space communications. RS erasure (RSE) codes are used in ATM networks and selected broadband applications [3], [4], [21], [37]. The Internet Engineering Task Force (IETF) recently proposed to standardize a new scheme [29], used for audio data in [7] and video data in [8], where a low quality copy of the data is sent as redundant data in subsequent packet(s).

A major difficulty when using FEC is to choose the right amount of redundant data in changing network condition scenarios, and it is usually not sufficient to only consider the mean packet loss ratio as for example outlined in Section II-B. The nature of the loss process has an impact on the recovery technique performance where different users in a multi-user environment experience different kinds of losses. Furthermore, a FEC scheme is only able to correct a certain amount of errors. In scenarios where a higher level of reliability is required, a *hybrid ARQ* scheme [13] might be appropriate combining FEC and ARQ. Type I hybrid ARQ is like a traditional FEC scheme, but if data is not recoverable, a retransmission is executed. Type II hybrid ARQ is like a traditional ARQ scheme, but parity data is first sent when a retransmission is required, and when data is not recoverable, the packet itself is retransmitted. Both hybrid ARQ types are currently not considered for the INSTANCE project, because they impose higher CPU time and memory demands onto the server than pure FEC to assure a degree of reliability that is higher than needed in INSTANCE. However, the suitab-

TABLE II
RS DECODING THROUGHPUT IN MBPS.

	no errors	max # errors	max # erasures
Scheme 1	3.620	1.772	1.394
Scheme 2	0.011	0.008	0.007

ility of these hybrid schemes for INSTANCE in large scale systems, e.g. NoD over the Internet, is subject to future research.

A.3 Summary of Existing Schemes

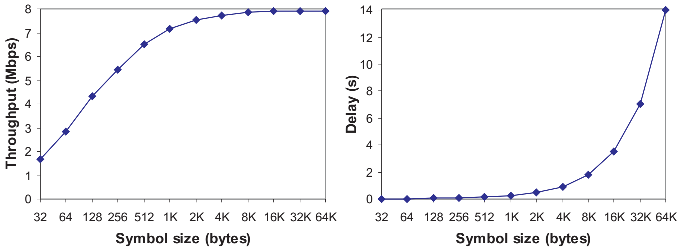
The potential of FEC to recover from losses is highly dependent on the loss behavior of the network [4]. Furthermore, the need of a disk array depends on required storage system performance and reliability. In our approach, it is necessary to protect the transmitted data stream against losses by adding a moderate level of redundancy using some kind of FEC and to increase performance, reliability, and load balancing of the disks in the storage system by using a disk array. Thus, an ARQ scheme is not appropriate, due to retransmission delays and the impossibility of integrating the mechanisms. The remaining solutions then use either an error and erasure correcting code, an erasure correcting code, or a combined error detection and erasure correction code. These three techniques are further evaluated by looking at reported studies of several available codes with appropriate correcting capabilities and by using Quantify (a software monitor) estimating the amount of needed CPU cycles on a 167 MHz Sun UltraSparc1.

B. Error and Erasure Correcting Codes

A straight forward solution is to use a correction code which corrects both errors and erasures. However, our experiments with these types of codes show that they do not meet our requirement number 1. Table II shows the measured throughput using an RS code. We introduced different amounts of losses and corruption, and we tested different code configurations able to restore the codeword if up to 12.5 % of the data is corrupted: (scheme 1) 1 B symbol, 255 symbols codeword and (scheme 2) 2 B symbol, 65535 symbols codeword. The main conclusion is that this code is inappropriate. Using a large codeword, the performance is the main bottleneck. Depending on the amount of damaged data and available CPU resources, the best data rate the code can deliver is 0.01 Mbps, i.e., no data is damaged. Using a small codeword, performance is still a problem where a data rate of 3.62 Mbps can be achieved during an error-free transmission, but due to the small codeword, a complex data interleaving must be introduced to scatter the codeword symbols. This is necessary to correct all the possible errors, because the amount of loss may exceed the small codeword size. Nevertheless, it gives an additional reorganization cost at the client side gathering the codeword symbols for decoding. As the results show, the decoding overhead is substantial irrespective of the amount of corrupted or lost data. Since we do not know whether there has occurred an error, we have to check and possibly correct the codeword. Furthermore, we have to send more redundant data in order to both detect and correct an unknown error.

C. Erasure Correcting Codes

Conventional FEC schemes like RS have a uniform error correction capability which can result in poor bandwidth utilization



(A) Decoding throughput. (B) Startup delay.

Fig. 3. Cauchy-based RSE decoding.

due to a complex algebraic decoding operation, and since we do not know whether the data is damaged or not, the decoding function must be run every time. Furthermore, as the bit error rate has dropped in wired networks, the packet loss rate due to random bit errors has been assumed negligible. Thus, congestion losses are often the dominant form of data loss, and the network can then be modeled as a well-behaved erasure channel where the unit of loss is a packet. Thus, by only using erasure correction, the recovery can be greatly simplified [3], and the correcting power can be approximately doubled [4].

Several erasure correcting schemes are developed, but not all are appropriate for our purpose. The traditional XOR based recovery of a stripe in a RAID system is not suitable, because of the missing ability to correct several random losses in the network. The idea of multi-dimensional parity calculations used in some disk array configurations to correct more than one disk failure, like RAID level 6 and as proposed in [5], [19], are not usable since the parity information corrects disk blocks spanning several different files, i.e., contradictory to requirement number 4. Moreover, the IETF scheme [29], using a low quality copy of the data for error recovery, is not suitable for our purposes, because it cannot handle disk failure and fails to meet requirement number 1.

RSE codes have the required capability to correct random packet losses in our scenario. Different implementations exist and are based on for example Cauchy matrices [6] and Vandermonde matrices [34] over finite fields. Since the Cauchy-based code is reported faster than the Vandermonde-based code [12], we have evaluated this code in the same way as the RS code above. This code is also more flexible with regard to choosing codeword and symbol size, so we were able to perform tests on more configurations of the code. The decoding performance in a worst-case loss scenario, i.e., 32 randomly selected packets out of 256 packets are lost, is illustrated in Figure 3. As the figure shows, the decoding throughput increases with the symbol size, but unfortunately, so does the experienced startup delay. Furthermore, the encoding times are approximately constant, but the decoding times may vary, i.e., increasing with the number of redundant symbols that are needed [6]. However, our measurements show that this code can give an adequate performance by choosing an appropriate codeword and symbol size.

Nevertheless, bit errors will also be a source of loss in future networks, especially in wireless networks. The above RSE codes are not able to correct unknown errors, and the received data may therefore have undetected and unrecovered errors. Hence, a solution may be to integrate different schemes which we look at in the next subsection.

D. Combined Codes

Possible solutions can be based on the fact that congestion (packet) loss is the dominant data corruption source. Therefore, a combination of the erasure correcting scheme described above with some kind of error correction or detection scheme can be used.

When combining the erasure code with an error correcting code, a fast, efficient inner error correcting code is first applied within each packet to correct bit errors introduced by noise. Then an outer erasure correcting scheme is used on a larger information block spanning several packets capable to correct packet losses. If an error is detected which cannot be corrected by the inner mechanism, or the packet is lost, the symbol represented in this packet is marked as an erasure where the outer recovery scheme is responsible for restoring the original data block. In such an integrated recovery mechanism, the outer erasure correcting scheme will work as the storage system recovery mechanism and correct failures due to bad disk blocks or disk crashes.

This scheme is able to correct the types of data corruption which we address in our scenario, but we introduce even more redundant data, and performance might still be a problem. However, as packet loss is the dominant source of data damage, we can replace the error correction scheme with a less CPU intensive error *detection* code. Thus, if an error is detected, the packet is only marked damaged and corrected by the outer erasure code. This approach seems to be the best solution, and in the next section, we show how it is realized in our prototype design.

IV. Prototype Design

Based on the results from our evaluation of some of the existing codes, we present our MoD server prototype, which is implemented in NetBSD, and especially the integrated error management mechanism. Since retransmissions are not suitable for our purposes, we use UDP as the transport level protocol. For data loss and corruption recovery, we use a combination of an outer Cauchy-based RSE correcting code and an inner error detecting code. To avoid introducing extra overhead and redundant data, we use the UDP checksum [11], which is calculated by dividing the whole packet, including header and pseudo header, in 16-bit blocks and adding each block using 1's complement sum, for error detection. Thus, if the UDP protocol finds a bit error in the received packet by re-calculating the checksum at the client side, it is marked as a lost packet and then corrected by the outer erasure code.

Our three techniques to improve performance in an MoD server can be used alone or combined in a system. The memory architecture has no direct implications on the integrated recovery scheme, but since we use the UDP checksum for error detection, the network level framing design is closely integrated with the recovery scheme. Hence, in the next subsections, we first describe the relation between the integrated error management scheme and network level framing. Then, we analyze the impact of the crucial parameters codeword size, symbol (and packet) size, and disk block size on the system performance and its recovery capability. In our current prototype, we actually use a single disk system and simulate a disk array of eight disks, i.e., seven information disks and one parity disk. However, the disk configuration has no effect on the resulting encoding and decoding performance and memory usage.

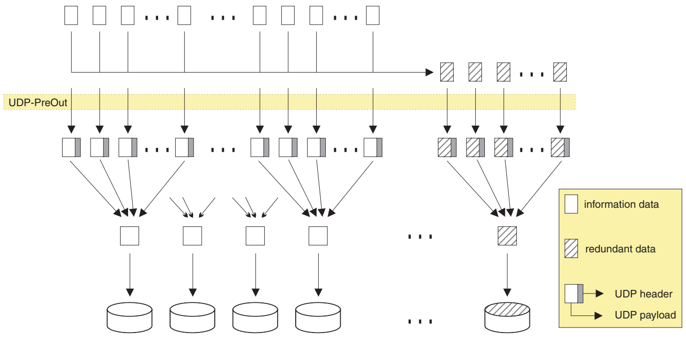


Fig. 4. Writing data to disk.

A. Integrated Error Management and Network Level Framing

To preprocess data through the UDP layer and store packets on disk, we split the traditional `udp_output` procedure [39] into two distinct functions: (1) `udp_PreOut`, where the UDP header is generated, the checksum is calculated, and the packet is written to disk, and (2) `udp_QuickOut`, in which the remaining fields are filled in and the checksum is updated with the checksum difference of the new header fields.

As depicted in Figure 4, write operations are done à la RAID level 4/5 where the information data is used to generate a set of redundant parity data for recovery, i.e., using the Cauchy-based RSE code. Each of the codeword symbols are then processed through `udp_PreOut` to prefabricate and store the UDP packet, i.e., the output of CPU intensive operations like the checksum calculation is stored in the packet header. Finally, to optimize storage system performance, several codeword symbols are concatenated to form a disk block, and the newly formed blocks are written to disk.

To reuse the stored parity data for FEC in the communication system, read operations are performed in a RAID level 0 fashion as sketched in Figure 5. The whole UDP packet is retrieved from disk containing either media data or parity data and processed through the `udp_QuickOut` where only a quick assembly of the packet is performed. Consequently, the server side UDP protocol operations are reduced to a minimum at transmission time (see Section IV-E).

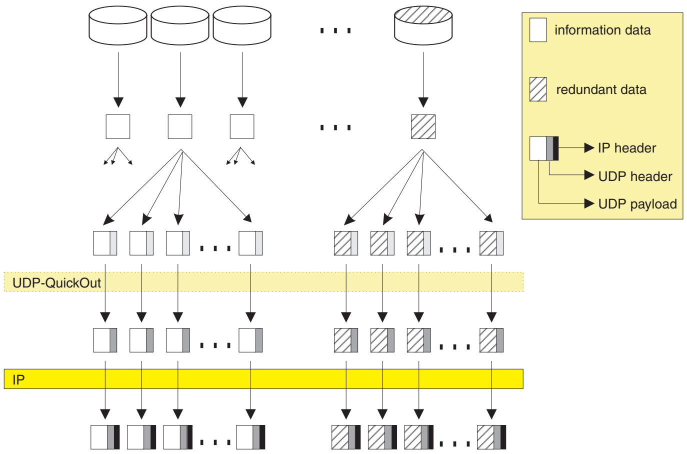


Fig. 5. Retrieving data from disk.

B. Codeword Configuration

The codeword should be stored in one or more whole stripes, i.e., codeword_size = stripe_size $\times$ n, n $\in \mathbb{Z}^+$, to avoid having data from more than one file within a codeword and to avoid fragmentation due to leaving some of the disk stripes empty. In our prototype approach, we use a codeword consisting of 256 symbols of which 32 symbols are redundant symbols. Thus, the codeword symbols are easily distributed among the striping units without any fragmentation. We can recover from one disk failure, and the code corrects most of the errors in the network, i.e., we can lose any 32 out of 256 packets which covers most of the network errors occurring in the packet loss reports described earlier.

C. Symbol Size

The size of the codeword symbol has impact on the decoding performance. Decoding throughput and additional introduced startup delay due to decoding is dependent on the codeword configuration, the symbol size, and the available CPU resources. To find suitable symbol size values, we use the worst-case evaluation results (see Section III-C) to see the behavior of the code according to the symbol size. Figure 3A shows that the throughput increases with larger symbol sizes. If we use symbol sizes equal or larger than 1 KB, the client system will be able to decode streams with data rates of 6 Mbps and more. However, increasing throughput by using larger symbol sizes also increases the startup delay (see Figure 3B), because there is more data to decode per codeword. Therefore, to reduce the experienced startup delay, we do not use symbol sizes larger than 8 KB. This corresponds to a decoding delay of 2 seconds or less.

D. Disk Block Size

In the storage system, the performance of an I/O operation is highly dependent on the disk characteristics. The disk throughput is dependent on the disk arm movement speed (seek time), the spindle speed (rotational latency and data transfer speed), the size of the transfer request, the scheduling of the requests, etc. Data is retrieved from disk in blocks which consist of multiple disk blocks (usually of 512 B each), and a striping unit in a disk array may similarly consist of a configurable number of disk blocks. Different applications may use different storage systems where striping units of varying sizes can be applied, i.e., sizes up to 64 KB are suggested for multimedia data whereas traditional systems usually use sizes of 4 KB or 8 KB.

Disk seeking and rotation latencies are main bottlenecks in disk I/O operations. Therefore, the larger the amount of data read per operation, the higher the throughput. In other words, increasing disk block size increases performance. However, some operating systems have an upper limit of the disk block size. Thus, in the full implementation of our server, we will use a block size, depending on the symbol size, of 32 KB (1 KB symbols) or 64 KB (2 KB, 4 KB, or 8 KB symbols). Nevertheless, it is important to note that in the performance evaluation below, the disk block size will not affect the evaluated code performance.

E. Precalculated Checksum

Before storing data in the RAID system, we process the data through the `udp_PreOut` function to generate UDP packets with (precalculated) checksums (see Figure 4). Originally, the UDP checksum is calculated over the three areas displayed in Figure 6: a 12 B pseudo header containing fields from the IP header, the 8 B UDP header, and the UDP data [39]. To use the prefabricated packet for transmission from server to client, we have to fill in the corresponding, i.e., future, source and destination IP addresses and port numbers. However, most of these values are unknown when storing the data. The source IP address will be the server's IP address, but the source port number will be allocated dynamically at transmission time and the client's destination IP number and destination port number are also unknown. Therefore, these unknown fields are initialized with zero before the checksum is calculated from the modified UDP sender.

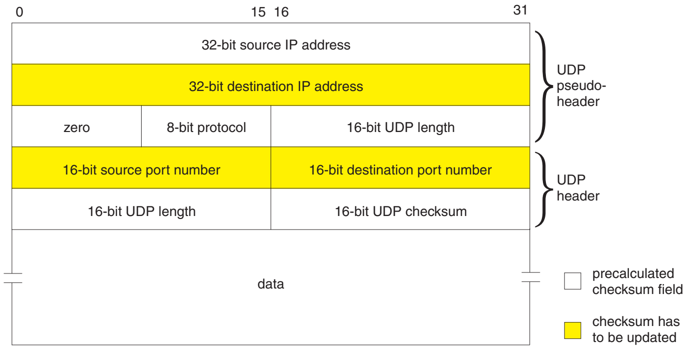


Fig. 6. Pseudo header, UDP header, and data used for checksum computation.

In order to transmit data from server to clients, the prefabricated UDP packets are read from disk and then processed through the communication protocols UDP/IP, but using `udp_QuickOut` instead of the traditional `udp_output` at the transport layer. In `udp_QuickOut`, only the following four steps are performed: (1) the missing values, i.e., source port number, destination port number, and destination IP address, are filled in the appropriate fields of the UDP packet, (2) a checksum over these three fields is calculated, (3) the value of the UDP checksum field is updated by adding the checksum of the three new field values to the precalculated UDP checksum, and (4) the UDP packet is handed over to the IP protocol.

The gain of the checksum precomputation depends on the packet size. In our scenario using four packet sizes of 1 KB, 2 KB, 4 KB, and 8 KB, updating only the source port (16 bit), destination port (16 bit), and destination IP address (32 bit), the on-line checksum procedure is executed over only 0.77 %, 0.39 %, 0.19 %, and 0.10 % of UDP data, respectively. This should again result in a throughput improvement similar[2] to the difference between the UDP measurements with and without checksum described below in Table III.

F. Prototype Configuration Summary

We have chosen and tested four different schemes using a correcting code with a codeword consisting of 224 information symbols and 32 parity symbols and where the symbol size is varied among 1 KB, 2 KB, 4 KB, and 8 KB. As depicted in Figure 4, we gather several symbols in one striping unit of 32 KB or 64 KB. In the next section, we present the performance measurements of our integrated recovery scheme using the Cauchy-

[2] It will not be exactly the same, because we still have to update the checksum. The checksum is not completely removed, but only reduced to a minimum.

based RSE code for storage recovery and FEC in the communication system.

V. Results and Observations

To analyze the performance of our prototype, we have designed a simple benchmark application in which we transfer a 225 MB file with 6 Mbps between the server process and the client process. A worst-case evaluation is made by introducing during transmission a maximum amount of errors within the correcting limits of the code, i.e., 32 out of 256 packets are damaged or lost. To test different server and client hardware, we have run our experiments on six different machines, i.e., three running as multi-user servers (A-C) and seven as "stand-alone" client machines (D-J): (A) Sun UltraSparc1 167 MHz, Solaris, (B) Sun Enterprise 450 (Ultra 4) 300 MHz, Solaris, (C) Dell PowerEdge 6300 500 MHz, Linux, (D) Dell Inspiron 7000 400 MHz, Linux, (E) Cinet PPI-600 350 MHz, Linux, (F) Tech Pentium 166 MHz, Linux, (G) Tech AMD K7 700MHz, Red Hat Linux, (H) Cinet PPI-600 350 MHz, NetBSD, (I) Dell Inspiron 7000 400 MHz, NetBSD, and (J) Cinet PPI-600 350 MHz, OpenBSD.

A. Server Side

The server side performance gain, integrating the error management mechanisms and reading the parity data from disk, is substantial. Storing and retrieving parity data from disk require neither extra storage space (compared to traditional RAID systems where one disk is already allocated for parity data) nor extra time for data retrieval (as the recovery disk is read in parallel with the original information data). Since we use FEC, there is no overhead managing retransmissions, and since the parity data is pre-calculated, all the encoding operations are omitted. This means that encoding performance is not a bottleneck, and the number of concurrent users retrieving data from the multimedia storage server is limited by the performance of the system components, e.g., bus, host-network interface, and disk I/O throughput. For example, on machine A, the measured encoding throughput is, of course highly dependent of available CPU resources, about 11.2 Mbps for all our evaluated schemes (using symbols of 1 KB, 2 KB, 4 KB, and 8 KB). This means that if encoding operations were to be done by the communication system, less than two 6 Mbps streams could be transmitted concurrently assuming that encoding performance is the bottleneck. Furthermore, Table III shows the maximum throughput measured in a gigabit environment, using *netperf*, between two machines using 32 KB send and receive buffers, a 1 Gbps network, a 350 MHz processor, and running OpenBSD 2.6-current (machine J). The measured FEC encoding throughput on this machine is about 16.4 Mbps, 17.6 Mbps, 16.2 Mbps, 15.0 Mbps respectively for our four different schemes. Thus, performing the encoding operations on the server side would greatly reduce the maximum throughput. The only overhead on the server side in our approach is 12.5% of increased buffer space and bandwidth requirement to hold the redundant data in memory and transmit it over the buses and the network.

B. Client Side

By reconstructing the lost data at the client side rather than having a retransmission, the clients may experience a better data presentation at the cost of increased memory and CPU usage.

TABLE III

Maximum measured throughput in Mbps on machine J.

	UDP		TCP	FEC
Packet size	**without checksum**	**with checksum**		**encoding**
4 KB	∼420	∼385	∼200	∼16
8 KB	∼450	∼405	∼210	∼15

TABLE IV

Measured average decoding throughput in Mbps.

Symbol	**Machine**									
size	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
1 KB	6.02	9.99	8.16	10.03	8.96	3.18	17.10	10.35	10.62	14.87
2 KB	6.51	10.71	7.43	9.27	9.22	3.31	17.40	10.71	10.05	16.28
4 KB	6.36	10.55	6.51	9.48	9.05	3.38	16.67	10.37	10.52	14.85
8 KB	6.20	10.49	6.47	9.59	8.89	3.25	16.53	10.12	10.66	14.22

The different configurations using symbol sizes of 1 KB, 2 KB, 4 KB, and 8 KB have a memory requirement of approximately 2 MB, 3 MB, 5 MB, and 9 MB, respectively. Thus, if the amount of physical memory available is less, the decoding throughput decreases due to data paging, but this will usually not be a bottleneck today.

The average decoding[3] performance on our different machines is displayed in Table IV where most of our experiments show that a 6 Mbps data stream, with maximum data loss, can be recovered in time at the client side, i.e., using similar hardware as in our machines. However, the 166 MHz machine F is not able to support a higher data rate than about 3 Mbps which indicates some minimum hardware requirements depending on the supported data rate.

Our measurements show that the decoding throughput varies over time according to available CPU time. According to Table IV, there are no large performance differences varying the symbol size between 1 KB and 8 KB, and all these configurations are adequate regarding throughput. The server machines share the CPU between several processes and users, and consequently, more variance is experienced in the throughput compared to the "stand-alone" machines. In general, there is a larger variance using a small symbol size in all our measurements. This is due to the size of each operation. Context switches and interrupts occurring during a decoding operation will constitute more of the total time per operation for small symbol sizes, and if they do not occur, correspondingly less amount of time is used per operation. Thus, smaller symbol sizes give a greater variation than larger symbols.

Table V shows the experienced startup decoding delays in our experiment. Due to the decoding cost, the client might experience an increased startup delay ranging from about 0.2 seconds to about 4.5 seconds (if a maximum amount of errors occurs) depending on the processor speed and the block size used. The delay increases with the size of the symbol, and since there are no large differences in throughput of the evaluated schemes, a symbol size of 1 KB or 2 KB is appropriate.

Finally, as there usually is some variance in the accessibility of the processor (unless some kind of reservation-based scheduling is provided), and thus in the decoding throughput, some

[3]The encoding throughput (and delay) is approximately the same as the decoding throughput (and delay). Therefore, it is not presented here due to space limitations.

TABLE V
MEASURED AVERAGE STARTUP DELAY IN SECONDS.

Symbol size	Machine A	B	C	D	E	F	G	H	I	J
1 KB	0.30	0.18	0.14	0.18	0.20	0.58	0.11	0.18	0.17	0.12
2 KB	0.56	0.33	0.49	0.40	0.39	1.11	0.21	0.34	0.36	0.22
4 KB	1.16	0.69	1.11	0.82	0.85	2.16	0.44	0.71	0.70	0.49
8 KB	2.45	1.47	2.25	1.54	1.63	4.50	0.95	1.45	1.38	1.04

client-side buffering should be provided. Nevertheless, despite all the overhead introduced on the client side, the recovery can be made in time assuming client hardware similar to the machines we used in our experiments. Thus, using our integrated error management mechanism, the server workload is greatly reduced, and the clients experience a smoother data playout, because no latency is introduced due to retransmissions.

C. High Loss Rates, Congestion, and Bandwidth Utilization

The main concern in this paper is the integrated storage and communication system error recovery mechanism in INSTANCE, and its usage in small and medium scale applications like hotels and city-wide pay-per-view entertainment. In this context, local area and metropolitan area networks are typically used for data transmission. Both network technologies are characterized by low error rates and large bandwidths. Thus, our error model assumption (Section II-B) is appropriate in our application scenario.

Obviously, in the context of the Internet, i.e., large scale systems, there are scenarios where our assumption might not be fulfilled where the loss rates might be too high, bursts of losses due to congestion might be too large, and bandwidth can be a scarce resource. How the INSTANCE principles apply in the presence of such error models is subject to ongoing and future work. In this subsection, we only give a brief discussion of possible solutions/approaches:

- *Parity data adaptation:* in an "error free" environment we may choose not to transmit the parity data, and thus, the probability of congestion is reduced.
- *Rate adaptation:* if the network workload reaches its capacity, increasing the probability of congestion, we start transmitting a lower quality and/or lower frame rate version of the data reducing the data rate of the transmission. In our case, with pre-generated transport level packets, this means that the data is stored on disk several times in different qualities.
- *Loss concealment:* the loss can be hidden from the receiver by some sort of reproduction of the lost data as for example described in [30]. This can for example be done by simply copying the previous data element or by interpolation using the surrounding data to produce a replacement for the lost packet(s).

The problem of insufficient bandwidth and congestion might be solved/reduced by applying the two types of adaptation, and if unrecoverable errors occur, loss concealment can be used to repair the damaged stream. However, some losses are in most cases tolerable in multimedia applications. Therefore, those applications that need full reliability have to use TCP or another reliable transport protocol anyway.

VI. DISCUSSION AND CONCLUSIONS

The INSTANCE server architecture is based on three orthogonal techniques: zero-copy-one-copy memory architecture, network level framing, and integrated error management. In this paper, we presented our design, implementation, and evaluation of the integrated error management. In particular, we described how error management mechanisms in storage and communication system can be integrated and which basic requirements have to be fulfilled in such a solution. Furthermore, we evaluated different design alternatives, described our prototype implementation, and evaluated the integrated error management in our prototype.

Our results show that the server workload is significantly reduced by the INSTANCE integrated error management, and a comparison of different schemes is presented in Table VI. In the error recovery enabled solutions, only our approach is suitable, because the server workload is reduced to a minimum, i.e., no retransmissions must be handled, no parity encoding operations are performed, and the checksum calculation operation is reduced to a minimum. UDP combined with FEC is useless for an MoD application due to the encoding overhead. ARQ-based transport protocols, like TCP, are suited for reliable point-to-point channels, but in case of point-to-multipoint connections, the overhead managing reliability will be too expensive for the server. These protocols are resource intensive and may cause high delay variations due to retransmissions. Furthermore, error recovery disabled solutions could be used, but in case of a loss, the end-user's playout quality is degraded. Thus, the INSTANCE scheme requires less crucial resources, i.e., memory, CPU, and bandwidth at the server, scales well for large multicast groups, and does not introduce delay variations, i.e., makes a smooth presentation at the client easier. In other words, more concurrent users can be supported by a single INSTANCE server with the same or better quality than in traditional server implementations.

It is worthwhile to note that our prototype is based on FEC schemes that are available as public domain software. There exist several faster correcting codes, e.g., the Tornado code [12], [20], that are commercially distributed and were not available for evaluation in our prototype. Obviously, these codes promise even higher performance improvements for the integrated error management in INSTANCE.

The design of the other two INSTANCE techniques, i.e., zero-copy-one-copy memory architecture and network level framing, is finished. Our ongoing work in INSTANCE is concerned with the implementation of these two techniques in our prototype.

ACKNOWLEDGMENTS

The authors would like to thank Minna Kaisa Juonolainen for her initial study on integrated error management issues. We are also grateful for the valuable comments and suggestions to this paper from Denise Ecklund, Ketil Lund, and Chuanbao Wang.

This research is sponsored by the Norwegian Research Council, the DITS program under contract number 119403/431 (the INSTANCE project).

REFERENCES

[1] 3com: *"Ethernet Packet Loss Reference"*, http://www.support.3com.com/-infodeli/tools/netmgt/temwin/temnt97/091293ts/ploss3.htm, September 1998

[2] Allman, M., Glover, D., Sanchez, L.: *"Enhancing TCP Over Satellite Channels using Standard Mechanisms"*, RFC2488, January 1999

[3] Almulhem, A., El-Guibaly, F., Gulliver, T. A.: *"Adaptive Error Correction for ATM Communications using Reed-Solomon Codes"*, Proceedings of the 1996 IEEE SoutheastCon, Tampa, FL, USA, April 1996, pp. 227-230

TABLE VI
COMPARISON OF DIFFERENT ERROR RECOVERY APPROACHES.

	Error Recovery			No Error Recovery	
	INSTANCE: NLF + FEC	**UDP + FEC**	**TCP**	**INSTANCE: NLF**	**UDP**
Bandwidth utilization†	*∼100%* Only checksum update.	*∼3-4%* Limited by encoding operation.	*∼47-48%* See footnote to Table III.	*∼100%* Only checksum update.	*∼100%* (without checksum) *∼90-91%* (with checksum)
FEC encoding delay	*No*	*Yes*, See Table V.	*No*	*No*	*No*
FEC decoding delay	*Yes*, See Table V.	*Yes*, See Table V.	*No*	*No*	*No*
Checksum delay	*∼1%*, See Section IV-E.	*100%*	*100%*	*∼1%*, See Section IV-E.	*100%*
Data redundancy	*Yes*‡	*Yes*‡	*No*	*No*	*No*
MoD suitable	*Yes*	*No*	*No*	*Partly*	*Partly*

†Compared to the *UDP without checksum* measurement in Table III.
‡The amount of redundant data is recovery capability dependent, i.e., 12.5% in our scenario.

[4] Biersack, E. W.: *"Performance Evaluation of Forward Error Correction in an ATM Environment"*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, May 1993, pp. 631-640

[5] Blaum, M., Brady, J., Bruck, J., Menon, J.: *"EVENODD: An optimal Scheme for Tolerating Double Disk Failures in RAID Architectures"*, Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA'94), Chicago, IL, USA, April 1994, pp. 245-254

[6] Blömer, J., Kalfane M., Karp, R., Karpinski, M., Luby , M., Zuckerman, D.: *"An XOR-Based Erasure-Resilient Coding Scheme"*, Technical Report TR-95-048, International Computer Science Institute (ICSI), The University of California at Berkeley, CA, USA, August 1995

[7] Bolot, J.-C., Fosse-Parisis, S., Towsley, D.: *"Adaptive FEC-based Error Control for Internet Telephony"*, Proceedings of the 18th IEEE Conference on Computer Communications (INFOCOM'99), New York, NY, USA, March 1999

[8] Bolot, J.-C., Turletti, T.: *"Adaptive Error Control for Packet Video in the Internet"*, Proceedings of the 1996 IEEE International Conference on Image Processing (ICIP '96), Lausanne, Switzerland, September 1996

[9] Bolot, J.-C., Vega-Garcia, A.: *"The Case for FEC-based Error Control for Packet Audio in the Internet"*, To appear in ACM/Springer Multimedia Systems, 1999

[10] Boyce, J. M., Gaglianello, R. D.: *"Packet Loss Effects on MPEG Video Sent Over the Public Internet"*, Proceedings of the 6th ACM International Multimedia Conference (ACM MM'98), Bristol, UK, Septeber 1998, pp. 181-190

[11] Braden, R., Borman, D., Partridge, C.: *"Computing the Internet Checksum"*, RFC 1071 (Updated by RFC 1141 and 1624), September 1988

[12] Byers, J. W., Luby, M., Mitzenmacher, M., Rege, A.: *"A Digital Fountain Approach to Reliable Distribution of Bulk Data"*, Proceedings of the ACM conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'98), Vancouver, Canada, August/September 1998, pp. 56-67

[13] Carle, G., Biersack, E. W.: *"Survey of Error Recovery Techniques for IP-based Audio-Visual Multicast Applications"*, IEEE Network, Vol. 11, No. 6, November/December 1997

[14] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., Patterson, D. A.: *"RAID: High-Performance, Reliable Secondary Storage"*, ACM Computing Surveys, Vol. 26., No. 2, June 1994, pp. 145-185

[15] Dresler, S., Hofmann, M.: *"Adaptive Error Correction to Support Heterogeneous Multicast Groups"*, Proceedings of 6th Open Workshop on High Speed Networks, Stuttgart, Germany, October 1997, pp. 169-174

[16] Ginier-Gillet, P., Di Minico, C. T.: *"1000BASE-T: Gigabit Ethernet Over Category 5 Copper Cabling"*, 3com, http://www.3com.com/-technology/tech_net/white_papers/503047.html, December 1999

[17] Halsall, F: *"Data Communications, Computer Networks and Open Systems"*, Fourth edition, Addison-Wesley, 1995

[18] Halvorsen, P., Plagemann, T., Goebel, V.: *"The INSTANCE Project: Operating System Enhancements to Support Multimedia Servers"* (poster), WWW-site for the 17th ACM Symposium on Operating Systems Principles (SOSP'99), Kiawah Island, SC, USA, December 1999

[19] Hellerstein, L., Gibson, G. A., Karp, R. M., Katz, H. M., Patterson, D. A.: *"Coding Techniques for Handling Failures in Large Disk Arrays"*, Algorithmica, Vol. 12, No. 2/3, August/September 1994, pp. 182-208

[20] Luby, M., Mitzenmacher, M., Shokrollahi, M. A., Spielman, D. A., Stemann, V.: *"Practical Loss-Resilient Codes"*, Proceedings of the 29th annual ACM Symposium on Theory of Computing (STOC'97), El Paso, TX, USA, May 1997, pp. 150-159.

[21] McAuley, A. J.: *"Reliable Broadband Communication Using a Burst Erasure Correcting Code"*, Proceedings of the ACM Symposium on Communications, Architectures and Protocols (SIGCOMM'90), Philadelphia, PA, USA, September 1990, pp. 297-306

[22] MPEG.org: *"DVD Technical Notes - Bitstream breakdown"*, http://mpeg.org/MPEG/DVD, March 2000

[23] Nahrstedt, K., Steinmetz, R.: *"Resource Management in Networked Multimedia Systems"*, IEEE Computer, Vol. 28, No. 5, May 1995, pp. 52-63

[24] Nonnenmacher, J., Biersack, E. W.: *"Reliable Multicast: Where to use FEC"*, Proceedings of IFIP 5th International Workshop on Protocols for High Speed Networks (PfHSN'96), Sophia Antipolis, France, October 1996, pp. 134-148

[25] Ousterhout, J. K.: *"Why Aren't Oparating Systems Getting Faster As Fast As Hardware?"*, Proceedings of the 1990 USENIX Summer Conference, Anaheim, CA, USA, June 1990, pp. 247-256

[26] Patterson, D. A., Gibson, G., Katz, R. H.: *"A Case for Redundant Arrays of Inexpensive Disks (RAID)"*, Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, June 1988, pp. 109-116

[27] Paxson, V.: *"End-to-End Routing Behavior in the Internet"*, Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'96), Palo Alto, CA, USA, August 1996, pp. 25-38

[28] Paxson, V.: *"End-to-End Internet Packet Dynamics"*, Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97), Cannes, France, September 1997, 139-152

[29] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J.-C., Vega-Garcia, A., Fosse-Parisis, S.: *"RTP Payload for Redundant Audio Data"*, RFC 2198, September 1997

[30] Perkins, C., Hodson, O., Hardman, V.: *"A Survey of Packet Loss Recovery Techniques for Streaming Audio"*, IEEE Network, Vol. 12, No. 5, September/October 1998, pp. 40-48

[31] Plagemann, T., Goebel, V.: *"INSTANCE: The Intermediate Storage Node Concept"*, Proceedings of the 3rd Asian Computing Science Conference (ASIAN'97), Kathmandu, Nepal, December 1997, pp. 151-165

[32] Plagemann, T., Goebel, V., Halvorsen, P., Anshus, O.: *"Operating System Support for Multimedia Systems"*, The Computer Communications Journal, Elsevier, Vol. 23, No. 3, February 2000, pp. 267-289

[33] Plank, J. S.: *"A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems"*, Software - Practice and Experience, Vol. 27, No. 9, September 1997, pp. 995-1012

[34] Rizzo, L.: *"Effective Erasure Codes for Reliable Computer Communication Protocols"*, ACM Computer Communication Review, Vol. 27, No. 2, April 1997, pp. 24-36

[35] Rorabaugh, C. B.: *"Error Coding Cookbook - Practical C/C++ Routines and Recipes for Error detection and Correction"*, McGraw-Hill, 1996

[36] Seagate, Disk Products, http://www.seagate.com/cda/products/-discsales/guide/, February 2000

[37] Srinivasan, V., Ghanwani, A., Gelenbe, E.: *"Block Loss Reduction in ATM Networks"*, Elsevier Computer Communications, Vol. 19, No. 13, November 1996, pp. 1077-1091

[38] Tanenbaum, A. S.: *"Computer Networks"*, Third edition, Prentice Hall, 1996

[39] Wright, G. R., Stevens, W. R.: *"TCP/IP Illustrated, Volume 2 - The Implementation"*, Addison-Wesley, 1995

[40] Yajnik, M., Kurose, J., Towsley.: *"Packet Loss Correlation in the MBone Multicast Network"*, Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'96), London, UK, November 1996, pp. 94-99