Modeling and Detecting Failures in Next-generation Distributed Multimedia Applications

D. Cotroneo (1,2), C. di Flora (1), G. Paolillo (2), and S. Russo (1)
(1) Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II
Via Claudio 21, 80125 - Napoli, Italy
{cotroneo,diflora,sterusso}@unina.it
(2) Consorzio Interuniversitario Nazionale per l'Informatica,
Via Diocleziano 328, 80100 - Napoli, Italy
generoso.paolillo@napoli.consorzio-cini.it

Abstract

In this paper we investigate dependability issues of next-generation distributed multimedia applications. Examples of such applications are autonomous vehicle control, tele-medicine, audio/video control, and Internet telephony. For these applications the quality of the delivered multimedia data is a critical factor. According to the ITU-T (working group SG 12), the quality of a multimedia service as perceived by end-users is defined by three parameters: delay, delay variation, and information loss. It is paramount to formalize the concept of a failure from the user's perspective. This paper defines the correctness of a multimedia service as a function of temporal distributions of the user-related parameters. It proposes a strategy for modeling and detecting failures of the considered applications. In particular, the detection process is based on error filtering functions. We show that the combination of threshold-based mechanisms is quite suitable for implementing an efficient detection strategy. We also evaluate the effectiveness of the proposed mechanism both by simulations and by experiments performed on a prototype. Such a prototype is tested with respect to a case study application, consisting of distributed tele control based on RTP/RTCP standard streaming protocols.

This work has been partially supported by the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) and by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)"

1 Introduction

The increasing interest in the so-called next-generation distributed multimedia applications [1], such as autonomous vehicle control, tele-medicine, audio/video control, and Internet telephony, is motivating new models and strategies for addressing dependability issues of such applications.

A great deal of research has been conducted on dependability issues of multimedia systems, as described in section 7. In recent years, research efforts have progressed along the following dimensions:

- providing fault tolerant multimedia server architectures, exploiting some form of redundancy (data on disks, disk servers, servers). These architectures mainly prevent system from the occurrence of crash failures;
- defining network protocols that enforce Quality of Service (QoS) specifications;
- developing QoS-enabled middleware platforms, to leverage the quality of multimedia services.

Figure 1 shows the conceptual model of distributed multimedia systems we assumed throughout this paper. It consists of 3 components: $i)$ the device which has to be controlled; $ii)$ the sender; and $iii)$ the receiver. The sender might physically reside either on the controlled device or on a fixed control host. The user controls the device by forcing device movements through the remote controller, based on multimedia data captured by the acquisition device. This schema is being used in most of critical multimedia applications, such as the most recent "TeleBAT" project [37].

A multimedia service, s_i, consists of a transmission of n packets, representing the media content, from the server

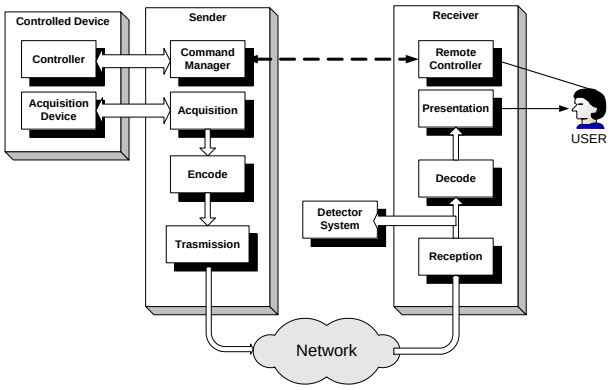

Figure 1. Conceptual model of next generation distributed multimedia systems

to the client. Service delivery encompasses the following phases: $i)$ media content acquisition (e.g., from a digital camera, from a disk); $ii)$ data encoding according to a particular format (e.g., JPEG, MPEG-1/2, H263); $iii)$ data transmission through a communication infrastructure using a standard streaming protocols, such as RTP/RTCP [29], DSM-CC [30], XTP [31]; $iv)$ data reception; $v)$ data decoding; and $vi)$ data presentation.

This paper focuses on failure modeling and detection of next-generation distributed multimedia applications. Although failure detection and modeling has been extensively studied, next-generation multimedia services raise a variety of issues, which have not been addressed before. Such issues stem from a number of factors, which are briefly described in the following.

First, quality attributes of next-generation multimedia applications are strictly related to the user perception. Suffice it to think of some kinds of critical distributed multimedia applications, such as tele-immersion and tele-medicine ones [2].

Second, investigating source of faults and providing fault-tolerance strategies for such systems is a quite complex task. In fact, a large number of factors may influence the quality of the delivered media. In particular, the geographic location of senders and receivers, the resource availability, the characteristics of the communication infrastructure, and other conditions which are not predictable at design time (e.g., load conditions, number of users).

Third, multimedia applications can tolerate transient errors and, under some circumstances, also intermittent and permanent errors. As an example, let us to assume an MPGE-2 audio/video transmission [4]. This is characterized by three kinds of frames: frame I, P, and B with a different weight on the quality of the media presentation. It has been demonstrated that frame B looses do not compromise the media quality at all, whereas frame I is the most important [3].

For the above mentioned reasons, in the context of critical multimedia applications, it is crucial that data about user-related quality parameters be carefully gathered and processed. In these systems, errors manifestations always result in the degradation of quality parameters as perceived form users. Our driving idea is to monitor and process such parameters as they are received, in order to detect failures. For instance, in tele-control applications, a failure occurs when multimedia data gets corrupted, being the user not able to control the device anymore. Recently, the International Telecommunication Union Telecommunication Standardization Sector (ITU-T) has started a studying group, namely SG 12, on the definition of the quality attributes of multimedia services from the user perspective. During the first meeting of the SG 12 group [8, 9], it has been pointed out the definition of the Quality of a multimedia Service as a function of three user-related parameters: i) **delay**, ii) **delay variation**, and iii) **information loss** (il).

Several research studies have demonstrated that these parameters are crucial for evaluating the quality of a multimedia service, as perceived from users [5, 6, 7]. Our goal is twofold: $i)$ defining the correctness of a multimedia service as a function of temporal distributions of the user-related parameters; and $ii)$ designing and implementing a failure detection mechanism. Several threshold-based mechanisms have been proposed and effectively applied to many fields (e.g. diagnosis, telecommunications, COTS-based distributed systems). Among these mechanisms, alpha-count has been recently proposed in [12] for discriminating transient vs. intermittent hardware faults. The driving idea is to inherit this mechanism in order to design a combination (a sort of alpha-count net) which is suitable for our objective, i.e., failure detection. **[N.B. ATTENZIONE A QUESTA FRASE: We demonstrate that this combination leads to an efficient schema for detecting failure based on the observation of user-related parameters.]** To this purpose, the detector system has to reside between the reception and decode modules. For this reason we plan to realize a detection module that is delivered along with a player component, as illustrated in section 5. We detail the description of implemented solutions with respect to a case study applications, consisting of a tele-control application based on RTCP/RTP standard streaming protocol [29].

The rest of the paper is organized as follows. Section 2 gives some preliminaries about the alpha-counts mechanisms. Sections 3 discusses the definition of the failure modes. Section 4 addresses the design issues of the detection system and presents several simulation results in order to make preliminary consideration concerning the configuration parameters of the detection system. Section 5 de-

scribes our case study-application and in Section 6 several experimental results conducted on the prototype are presented. Section 7 discusses pros and cons of our approach, as compared to several previous relevant works. Section 8 concludes the paper with final remarks and lessons learned.

2 Preliminaries

As far as the detection process is concerned, we investigate the applicability of threshold-based functions. In fact, among the heuristics based on the concept of threshold, the alpha-count family of mechanisms seems to be particularly interesting for our purposes due to effective and simple analytical characterization and to the thorough analysis already conducted. The basic alpha-count mechanism is a count-and-threshold scheme originally devised "to consolidate identification of faults, distinguished as transient or permanent/intermittent" [12]. The alpha-count mechanism is simple enough to: i) be implementable as small, low-overhead and low-cost modules, suitable even for embedded real-time systems, and ii) allow its behavior and its effects on the system to be explored using standard analytical means. An error signalling mechanism is assumed to collect error signals from any error detection device in the system. Error detection results are delivered, as binary signals, to the alpha-count mechanism on a regular time basis. The judgement on a component's behavior, given by the error signaling mechanism, is correct with a probability, or coverage, c. The alpha-count processes information about erroneous behavior of each system component, giving a smaller weight to error signals as they get older. A score variable α_i is associated to each not-yet-removed component i to record information about the errors experienced by that component. α_i is initially set to 0, and accounts for the L-th judgement as follows:

- $\alpha_i(L) = \alpha_i(L-1) + 1$ *if component i is perceived as faulty during execution i*
- $\alpha_i(L) = K * \alpha_i(L-1)$ *if component i is perceived as correct during execution i*

where $0 < K < 1$.

When $\alpha_i(L)$ becomes greater than or equal to a given threshold α_T, component i is diagnosed as failed and a signal is raised to trigger further actions (error processing or fault treatment). The effectiveness of the mechanism depends on the parameters K and α_T. The optimal tuning of these parameters depends on the expected frequency of errors, and on the probability c of correct judgements of the error signaling mechanism. The analysis performed in [12] has clearly shown the trade-offs between delay and accuracy of the diagnosis and thoroughly discussed ways to tune these parameters for optimizing mechanism's behaviour.

3 Failure Modes

A multimedia service is characterized by the following three temporal distributions:

1. *delay distribution*, contains a sequence of values $t_d s_i(n)$, which represent the delay of the n-th packet. If the packet is dropped by communication links, the delay value is greater than the maximum admissible, namely D_{max};
2. *delay variation distribution*, contains a sequence of values $t_{dv} s_i(n) = t_d s_i(n) - t_d s_i(n-1)$, which represent the delay variation of the n-th packet;
3. *information loss distribution*, contains a sequence of values $t_{il} s_i(n)$, representing the percentage of packets that have been delayed more than D_{max} in an observation period T. Such values are calculated as follows: $il_i(n) = \frac{\sum_{j=n-T}^{n} t_{pl} s_i(j)}{n}$, being

$$t_{pl} s_i(j) = \begin{cases} 1 & \text{if } d_i(n) > D_{max} \\ 0 & \text{otherwise.} \end{cases}$$

We define the failure modes according to the definitions given in [10]. We do not consider value errors throughout this paper. Such errors occur when data gets corrupted while traversing from the server to the client. Multimedia applications can tolerate such a kind of errors under many circumstances [11]. Let STD_i and $STDV_i$ be the subsets of all delay and delay variation distributions, respectively, which do not compromise the quality of the multimedia content. The definition of these sub-sets is purely theoretical and they would be perceived by an omniscient observer. We can define four classes of failures:

- `delay failure`, f_d, occurs when the following condition holds: ($t_d s_i \notin STD_i$). It is worth noting that a `crash failure` is a particular case of the delay failure. Indeed, a crash failure occurs when ($t_d s_i \in STCF_i \subset \neg STD_i$), being the $STCF_i = \{\exists i : \forall j \geq i, t_d s_j(j) \geq D_{max}\}$;
- `delay variation failure`, f_{dv}, occurs when the following condition holds: ($t_{dv} s_i \notin STDV_i$).
- `information loss failure`, f_{il}, occurs when ($t_{il} s_i \geq q_{il}$), being q_{il} the maximum admissible percentage of lost packets for the service s_i;
- `transparent failure`, f_{trp}. During the experiments performed on our prototype, based on tele-control application(see section 5), we noticed that there are some circumstances where all values

Media type	Application	Bandwidth	Quality Parameters		
			Delay	Delay variation	Information loss
Audio	**Conversational**	4-64 kb/s	<150msec <400msec	<1msec	<3% PLR
Audio	**Messaging**	4-32 kb/s	<1 s <2 s	<1msec	<3% PLR
Audio	**High quality audio Streaming**	16-128 kb/s	<10 s	<1msec	<1% PLR
Video	**Videophone**	16-384 kb/s	<150msec <400msec	-	<1% PLR
Video	**One-way**	16-384 kb/s	<10 s	-	<1% PLR

Figure 2. Multimedia Application requirements

$t_d s_i, t_{dv} s_i, t_{il} s_i$ change at same time. This effect compromised the quality of the multimedia content in that the user was not able to control the device anymore. Based on these observations we defined a new class of failure, namely transparent failure. Being $np_i = (t_d s_i, t_{dv} s_i, t_{il} s_i)$, i.e., np_i is a point of the three dimensional space $CV = STDT_i \times STDVT_i \times \{q \in R : 0 \leq q \leq q_{il}\}$, where $STDT_i \subset STD_i$ and $STDVT_i \subset STDV_i$. A transparent failure occurs when $np_i \in CV$.

We can define the correctness of a multimedia service as follows: *a multimedia service is defined to be correct if none of the above mentioned failures occurr*. We believe the above failures encompass all failures that can affect a multimedia system.

4 Detection system

In order to design the detection system, a number of key issues have been addressed. In particular, being STD_i and $STDV_i$ not computable in practice, we evaluate two alternatives: i) to find an heuristic strategy which attempts to individualize STD_i and $STDV_i$; ii) to choose an efficient and configurable schema which is able to detect failures with a relatively high coverage. Due to the variable characteristics of multimedia contents and environmental conditions, we claim that the first alternative is not realizable. We thus investigate the design and the implementation of a schema based on error filtering functions. Errors occur when values $t_d s_i(n)$, $t_{dv} s_i(n)$, and $t_{il} s_i(n)$ do not satisfy application requirements. Indeed, as suggested by the ITU-T and represented in Figure 2, multimedia applications requirements consist of three values d_{spec}, dv_{spec}, and il_{spec}, which represent the maximum values for delay, delay variation, and information loss, respectively.

Our driving idea is to monitor and process such parameters as they are received, using a combination of alpha-count functions. These functions can be configured in order

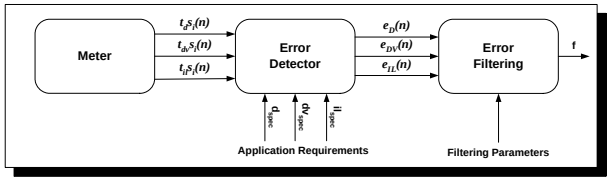

Figure 3. Conceptual model of the detection system

to find a trade-off between the accuracy and the detection time, as described later in this section. Figure 3 depicts the conceptual model of the proposed detection system. It consists of three subsystems:

- `Meter`, which is in charge of measuring the quality parameters for each packet;
- `Error Detector`, which has the responsibility of detecting errors. This subsystem is able to detect three kinds of errors: i) *error on delay*, defined by a boolean value $e_D(n) = (t_d s_i(n) > d_{spec})$; ii) *error on delay variation*, $e_{DV}(n) = (t_{dv} s_i(n) > dv_{spec})$; and iii) *error on information loss*, $e_{IL}(n) = (t_{il} s_i(n) > il_{spec})$;
- `Error filtering`, which is in charge of filtering the incoming error signals in order to detect failures.

The error filtering subsystem exploits a combination of threshold-based mechanisms based on alpha-count [12]. In order to design such a subsystem, we address the following two issues, separately: i) detecting failures f_d, f_{dv}, and f_{il}, i.e. failures which stem from the degradation of one parameter; and ii) detecting failure f_{trp}, i.e. the defined *transparent failure*. In order to address the first issue, we use three separate alpha-count functions, TBf_d, TBf_{dv}, and TBf_{il}, each providing detection of failures on a single parameter.

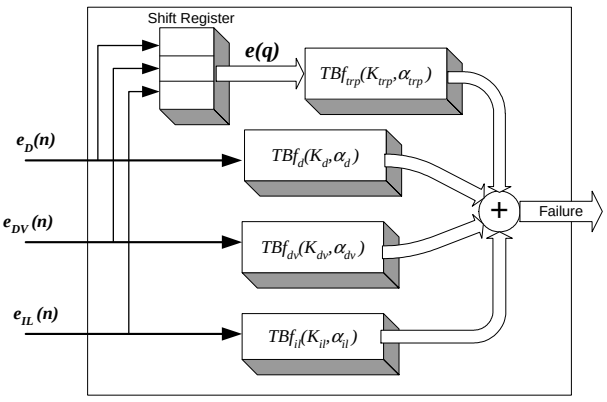

Figure 4. Error filtering subsystem

By tuning these functions it is possible to configure the accuracy and the detection time of each single detector. In order to detect transparent failures we use an alpha-count function, TBf_{trp}, which receives all the error signals at all. We exploit a parallel-in, serial-out register in order to stimulate the function with the error signals. It is worth noting that this alpha-count works with a frequency three times greater than the one of the input signals (for a given n, this function receives three signals e). As a result, we design an error-filtering subsystem which stems from the combination of the two mentioned solutions, as illustrated in Figure 4.

As defined in [14], two are the metrics which characterize a failure detector: i) the detection time (how fast the failure detector detects failures), and ii) the accuracy (how well it avoids mistakes). In the following we discuss preliminary simulation results obtained by modeling the error filtering subsystem through the MATLAB environment [33]. We use experimental data, representing a real multimedia content, as inputs for the model. Data are stored in files with different multimedia formats (see Figure 8 in the next section). We implemented three filter modules, using the MATLAB programming language, in order to generate delay, delay variation, and information loss distributions artificially. Two are the outputs of the simulation models: the failure signal, which is true if a failure has been detected, and the filtered multimedia data. In particular, we use the latter for evaluating the quality of the multimedia data when a failure has been raised. In this way, we can heuristically evaluate the accuracy of the fault detector. To this purpose, we developed a multimedia player module that receives the failure signal, as produced by the model, and the filtered multimedia data. This module automatically presents multimedia data for a time interval where the failure has been detected. This allows us to check if a false detection occurs and to perform a better tuning of the filtering parameters. The configuration of the alpha-count TBf_{trp} is not straightforward, in that it depends on the characteristic of multimedia format. Indeed, multimedia applications can be characterized by a Constant Bit Rate (CBR) or Variable Bit Rate (VBR) flows. One drawback of using VBR sources is the increased possibility of packet loss. This is due to high peak-to-mean ratios and significantly high autocorrelations, which characterize these sources. Such sources can be deleterious to networks, since they can cause high values for delay variation if network resources are not properly allocated. Using VBR flows, a transparent failure occurs when $t_{dv}s_i(n)$ and $t_{il}s_i(n)$ degrade at same time, i.e., for a given packet n, $e_{dv}(n) = e_{il}(n) = 1$. CBR flows allow to simplify bandwidth allocation, and also render the video source more amenable to traffic policing. However, CBR encoding has the drawback that video quality (distortion) varies significantly in order to maintain a constant bit-rate. Using CBR, a transparent failure occurs more likely when

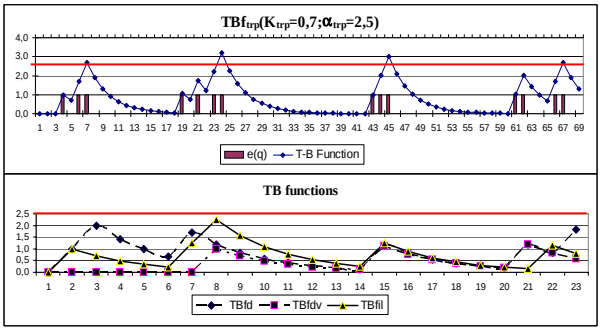

Figure 5. Transparent failure occurrences

all the three parameters degrade at same time. In order to better validate the foregoing considerations, we simulated the model by generating VBR (CBR) sources directly, according to the statistical distributions given in [15], and we obtained mainly the same results.

Figure 5 reports outputs of the alpha-count functions in the case of VBR video streaming. Four transparent failure have been raised by the $TBftrp$. We use value 0.7 for Ks and 2.5 for thresholds α. It is worth noting that the video quality is actually compromised for those packets corresponding to the raised failures.

We investigate the configuration of the filtering parameters. The configuration of the alpha-counts is performed according to the suggestions given in [12], where an exhaustive analysis of these parameters has already been performed. Simulation results indicated that, for the considered multimedia applications, lower values of k are needed. In fact, K represents the time window where memory of previous errors is retained; choosing k between 0.7 and 0.8 results in a time window which is large enough to keep track of previous errors without slowing too much the decrease of the alpha function.

5 The case study application

5.1 Overall architecture

This section describes the case-study application, consisting of real-time control of a Lego Mindstorm [16] robot. In particular, we developed an application for controlling the so-called “RoverBot”, described in the Lego MindStorms “Constructopedia”. The RoverBot Control (RBC) application enables remote users to receive a live videostream from the robot, and allows them to send movement-commands to the RoverBot. We adopted the Jini technology because of its code-mobility features, which allowed us to test the mechanism on different platforms and client devices [32]. A Jini service consists of a server-side back-end and a client-side *service object*. In compliance with the Jini

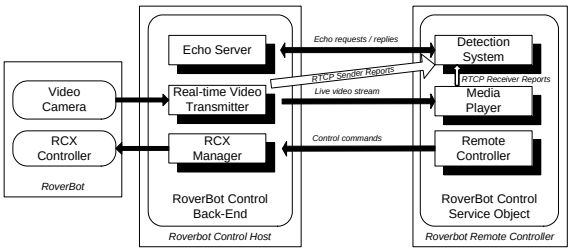

Figure 6. Overall architecture of the case-study application

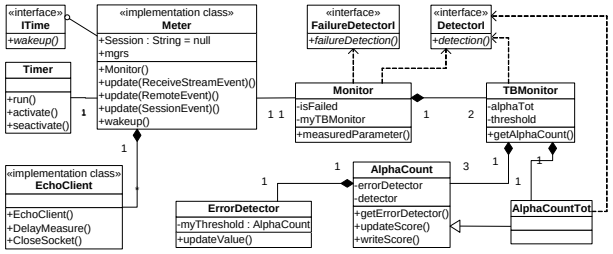

Figure 7. UML class diagram of the detection system

service model, the RBC has been split between the client side, i.e., the remote-driver side, and the back-end side, i.e., the Roverbot side. The overall architecture of the presented case-study application is depicted in Figure 6.

RBC's back-end consists of the following components: $i)$ the *real-time video transmitter* which produces a live video stream representing RoverBot's viewpoint; such a stream is based on the real-time transport protocol (RTP/RTCP) [29]; $ii)$ the *RCX manager*, which interfaces RBC back-end with RoverBot's RCX controller, in order to force it to perform specific movements; and $iii)$ the *echo server*, which is in charge of answering to echo requests sent by the *service object* for estimating the actual round trip time.

RBC's service object is composed by $i)$ the *media player*, which is in charge of presenting the video stream on remote driver's display, $ii)$ the *remote controller*, which allows the driver to communicate with the RBC back-end in order to move the RoverBot, and $iii)$ the *detection system*, which implements the mechanism described in Section 4 with respect to an RTP-based streaming session.

Multimedia-related components and part of the detection system have been here implemented exploiting the Java media framework (JMF) libraries [28]. The *RCX manager* is based on the RCX Java API [17] for communicating with the RoverBot.

Audio/Video	Format/Frequency[*Hz*]	Packet size [*byte]*	# Packets/sec. (*on average*)
Audio	**DVI/8000**	284	16
Audio	**DVI/11025**	374	16
Audio	**DVI/22050**	704	16
Audio	**ULAW/8000**	520	16
Audio	**GSM/8000**	139	16
Audio	**MPEG Layer III**	1298 /1141	12
Video	**JPEG/90000**	1008	(352x288) 220
Video	**H263/90000**	550 (*on average*)	(352x288) 70
Video	**MPEG-I/90000**	1000 (*on average*)	(352x288) 80

Figure 8. RTP packet sizes as a function of media formats

5.2 Implementation of the detection system

We choose to implement the detection system in a single component. For this reason we did not address issues concerning a distributed implementation of such a component. However, the interested reader may refer to [13] for a thorough description of a distributed implementation of threshold-based diagnostic mechanisms. Figure 7 shows the class diagram of the implemented detection system.

The *meter* class is responsible for estimating delay, delay variation and information loss. Information about the quality of the actual multimedia sessions is retrieved from RTCP session messages [29], called RTCP reports, which are periodically exchanged between senders and receivers. The observation period varies from 0 to 5 $secs$. Such messages provide measures about $i)$ the *interarrival jitter*, i.e., an estimation of statistical variance of RTP data packets' inter-arrival time, and $ii)$ the *fraction lost*, i.e., the fraction of packets that have been lost since the last report. The implemented *meter* uses such information to measure delay variation and information loss. Since the *interarrival jitter* is measured in $msec \times SamplingTime$, delay variation is calculated by dividing the *interarrival jitter* by the bit-rate (coding/decoding frequency). RTCP reports do not provide any information for estimating the delay. To this aim we provide the *Meter* with a particular class, *EchoClient*, which is in charge of calculating the round trip time (RTT) between the RBC host and the client. More specifically, the *EchoServer* is launched on the RBC host to estimate the delay between the two components. During the experiments we conducted on the prototype, we noticed that the delay estimation accuracy depends on the size of RTP packets. In order to improve the accuracy of such an estimation, we measured the size of RTP packets, transmitted using JMF libraries, for a large variety of formats. Results are shown in Figure 8.

During the start-up phase, the *EchoClient* retrieves the

format of the media content by analyzing RTCP reports, thus adapting the size of echo packets to the effective size of RTP ones. Each periodic evaluation of the RTT results from the mean value calculated on multiple measures; evaluations are periodically executed according to the wake-up messages provided by the *timer* thread. We configured the wake-up period at the same value of the observation one.

The *Monitor* class is in charge of triggering a failure signal once a failure is detected. Such a class instantiates two or more *TBMonitor* objects. Indeed, a multimedia content is composed by multiple flows (e.g., the audio and video flows). The *TBMonitor* class implements the error filtering subsystem, i.e., it is in charge of detecting failures for the assigned flow. The *AlphaCount* class implements the alpha-count function. According to the conceptual model, described in section 4, the *TBMonitor* is composed of 4 *AlphaCounts*. In particular, the alpha-count which implements the TBF_{trp} function, called *AlphaCountTot*, is a specialization of the *AlphaCount*.

6 Experimental results

In real-world scenarios, measures about quality parameters are performed in an observation period T. In fact, measuring these parameters for each packet, leads to solution not realizable in practice, characterized by an unacceptable overhead. For this reasons, standard streaming protocols, such as RTP [29], DSM-CC [30], and XTP [31], suggest an observation period, for monitoring actions, of $1 - 2[sec]$ on average. If a permanent error on a single parameter occurs, the detection time can be estimated as follows: $dt[sec] = \frac{\alpha}{K} \times T$.

We conducted experiments using the implemented case-study application to analyze and evaluate the behavior of the proposed detection mechanism. Experiments have been performed on a testbed, composed of a remote-control workstation (RCW), and a control host (CH) which effectively resides on the RoverBot, distributed over an IEEE 802.11b wireless LAN.

The RCW is a uniprocessor Linux machine running Linux Red Hat 8.0 distribution. The CH is a Sony Vaio Picturebook laptop, which is IEEE 802.11b compliant and it is equipped with an embedded video camera. We considered a real-time control application with the following QoS requirements: $i)$ d_{spec} = 80 ms, $ii)$ dv_{spec} = 10 ms, $iii)$ il_{spec}= 3%. The aim of out experimental evaluation was twofold.

Firstly, we conducted experiments to validate considerations and tuning rules given in Section 4. More specifically, given the specification of the QoS required by real-time control applications, we evaluated the suitability of variable bit-rate and constant bit-rate encoding. We tested both strategies for two kinds of video images, captured by a fixed video camera and a moving one. The former is typical of telemedicine applications, whereas the latter represents unmanned vehicle control applications. We used the H263 protocol for VBR streaming and JPEG for CBR streaming, with a video resolution of 352x288 pixels. Measures have been performed with a period $T = 2s$; hence, from Figure 8 it follows that each measure refers to 140 packets if the H263 protocol is used, whereas each JPEG-related measure refers to 440 packets. Experimental results showed that delay variation is substantially influenced by $i)$ the kind of images, i.e., coming from fixed camera or with rapid scene changes, and $ii)$ the encoding strategy, i.e., VBR or CBR. In particular, using H263 protocol resulted in an higher average of the delay variation, compared to JPEG: for a fixed camera, we obtained $t^{fix}_{dvH263} = 3.73msec$ and $t^{fix}_{dvJPEG} = 0.73msec$. Moreover, if rapid scene changes occur, we obtain $t^{mov}_{dvH263} = 9.0msec$ and $t^{mov}_{dvJPEG} = 0.88msec$. It is worth pointing out that delay variation in JPEG CBR encoding does not significantly depend on the kind of images; hence, upon scene changing rapidly, the resulting video is quite distorted. This confirms considerations we gave in Section 4.

Secondly, we conducted experiments to evaluate the suitability of the implemented mechanism with respect to failure modes defined in Section 3. Although it has been used for a critical application, the considered multimedia service can tolerate degradation of user-related parameters, until the perceived QoS allows the driver to control the remote device. Hence deciding whether a degradation leads to a failure or not, is a challenging issue that can be addressed by empirically evaluating the received stream. Experiments described in the following effectively validate the suitability of threshold-based filtering for distinguishing acceptable degradations from service failures. It is worth mentioning that we did not take into account delay failures because experiments showed that delay values were very low compared to service requirements, whereas we produced delay variation, information loss and transparent failures. Errors e_{DV} were generated by introducing a CPU-bound load on the control host. This load resulted in an increase of the delay variation, thus forcing jitter errors to occur. The second have been forced by injecting link faults on the ethernet switch that resides between the wireless LAN access point and the RCW. To do this we developed an application for enabling and disabling switch's port at run-time through a serial cable. In order to reproduce a transparent failures, we forced multiple light degradations of single parameters. In particular, we degradated $i)$ delay variation by generating a CPU-bound load which resulted in an 80% CPU occupation, $ii)$ information loss by disabling the switch for $500msecs$ according to user-defined distribution, and $ii)$ delay by executing multiple streaming sessions on the same CH. Figure 9 depicts detected errors and failures with re-

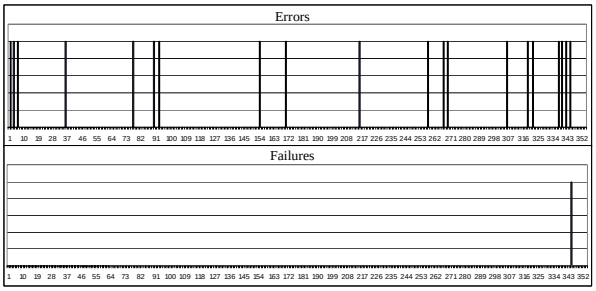

Figure 9. XXXX Detected errors and failures XXXXX

spect to about 700 seconds – i.e. 350 measures – of live streaming captured by the camera on the RoverBot. The detector raised a single transparent failure signal, due to errors detected between measures 330 and 350; moreover we generated several delay variation errors and information loss errors between measures 1 and 330. Experimental results show that only significant degradations of user-related parameters resulted in a transparent failure. The behavior of the failure detector is consistent with our empirical evaluation of the received images. We controlled the device until the transparent failure is raised, although several errors have been detected until measure 330. Subsequently, we were not able to control the RoverBot anymore. As far as crash failures are concerned, these are detected by the TBf_d functions.

7 Related Work

A great deal of research has been conducted on failure detectors. We recognize that the work presented in [14] represents a fundamental milestone for the pursuit of defining metrics for evaluating the quality of a failure detector. This paper defines two QoS metrics which quantify $i)$ how fast a failure detector detects actual failure, and $ii)$ how well it avoids false detections. The proposed failure detection algorithm has been provided under several assumptions: authors consider a message-passing distributed systems in which processes may fail by crashing and messages may be delayed or dropped by communication links. These assumptions are too restrictive with respect to our context of distributed multimedia systems. However, we defined in section 4 the outlined QoS metrics for the applications we considered.

As far as the user-related parameters are concerned, the work in [34] analyzes the perceived QoS referring to the same parameters. A new metric for video QoS is described in [36]; the proposed metric takes into account both the user and the service provider, in terms of user satisfaction and resource consumption respectively. The work relates to our definition of correctness for it defines user satisfaction in terms of the same parameters. It is more targeted to the definition of optimal settings for providing users with a certain QoS, rather than to the on-line estimation of QoS for failure detection. In [35] an experimental evaluation of the effects of CPU and network load on information loss and delay variation is proposed; in particular, this work points out the need of considering the temporal distribution of jitter and loss errors, emphasizing the effects of bursty errors on the perceived quality of service.

A number of solutions have been proposed in literature, that use different approaches for the realization of distributed services with real-time requirements. In particular, three main approaches have been here identified and described in the following. To the best of our knowledge, although the described research studies focused on dependability of distributed multimedia systems, none of them tried to formalize the concept of a failure from the user's perspective.

The first approach is based on the presence of specific services for the synchronization of components, such as the *timely computing base (TCB)* [18]. This approach is not suitable for distributed multimedia services for it does not take into account multimedia-related temporal parameters, such as the *delay variation*, as well as non-temporal parameters, such as the number of lost packets.

An alternative approach has been adopted by a variety of middleware-based solutions for addressing QoS requirements of real-time applications [19, 20, 21, 22]. In particular, in [19] a middleware is proposed, which allows applications to adapt communication protocols to the required QoS at run-time. Such a middleware, called *Da CaPo++*, enables the creation of communication flows that satisfy the specified quality attributes. This solution does not enable to handle transient degradations of service parameters. Moreover, QoS requirements may not change during application life-time. A more flexible solution is proposed in [20], which describes a mechanism for specifying elastic QoS requirements that can be adapted to the availability of network resources. This work focuses on the off-line establishment of real-time communication channels, and on run-time message scheduling. Such a mechanism might be used for checking correctness of multimedia services, even though no practical examples of such a usage are provided. The proposed mechanism is modeled as a Markov chain and evaluated trough simulation, whereas it has not been implemented or tested in a real environment. The work in [21] describes the design and the implementation of real-time dependable (RTD) channels, a communication-oriented abstraction that can be configured to meet the QoS requirements of a variety of distributed applications. A specific RTD channel is characterized by a set of service proper-

ties, such as real-time, reliability, ordering and jitter control. Such properties are specified in terms of probabilistic parameters during establishment of the channel, and cannot be changed at run-time. Customization of the QoS is enabled by the underlying *CactusRT* middleware, which estimates resource occupation exclusively trough its own information. This implies that if an external entity overloads system resources, the channel will not have a consistent view of the effective resource availability. The foregoing works provide solutions which are not specifically targeted to real-time multimedia systems. TAO is an open-source real-time CORBA ORB [22], which includes an implementation of the CORBA audio/video streaming (AVS) service [23]. This service provides applications with interfaces to specify and modify QoS, whereas the effective implementation of strategies for obtaining a specific QoS is delegated to applications.

The TAO ORB has been also used to implement a real-time video-streaming service for controlling an unmanned aerial vehicle [24]. This work defines an alternative concept of service correctness, based on three load conditions (e.g., low, medium, and high load), each associated to a specific frame-dropping level.

The third approach characterizes research projects that have been conducted in order to ensure correctness of multimedia systems by means of specific transport protocols. The work in [25] presents the multimedia enhanced transport system (METS): METS is an adaptive transport system that incorporates a QoS oriented API and a range of mechanisms to assist applications in exploiting QoS and adapting to fluctuations in QoS. Unfortunately, METS' portability is limited, being it based on multicast on a multi ATM switch network environment. In [26] an end-to-end transport architecture for multimedia streaming over the Internet is proposed, along with a new multimedia streaming TCP-friendly protocol (MSTFP), which combines forward estimation of network conditions with information feedback control to optimally track the network conditions. The proposed solution considers network conditions in terms of packet loss, which represents only one of the fundamental parameters we took into account. The work in [27] has the same limitations of [26], even though it also presents a strategy for fault forecasting and recovery.

8 Conclusions and Future Work

This work presented an approach for modeling and detecting failures in next-generation distributed multimedia applications. Due to the system complexity and not-predictable environmental conditions, we claimed that an effective approach for detecting errors is to monitor the errors manifestations in the form of quality degradations. Our driving idea was to define the correctness of multimedia as a function of user-related quality parameters in order to realize an efficient mechanism for detecting failures. To this aim, we used a combination of alpha-counts functions. Experiments on a real-world applications, suggested us to take into account a new class of failure that occurs when two or more quality parameters degrade at same time. In order to evaluate our strategy and the dependency with different multimedia formats, we developed a simulation model. We detailed the description of implemented solutions with respect to a case study applications, consisting of a tele-control application based on RTP/RTCP standard streaming protocols. Experimental campaigns were conducted referring to two scenarios: tele-medicine and autonomous vehicle control. Experimental results demonstrated that the considered applications can tolerate errors until the quality is compromised, i.e., the user is not able to control the device anymore. Honestly, we recognize that experiments are still preliminary to formalize and evaluate the quality metrics of the detector. However, our effort has been progressed from the formalization to the implementation issues, providing proofs of the realizability of the proposed mechanisms. We claim that this approach is a novel one for coping with dependability issues of next generation multimedia applications. Future work will aim at: $i)$ massive simulation and fault-injection experiments in order to better characterize the relationship between the filtering parameters and quality parameters of the detector system; $ii)$ to implement a recovery mechanism which is triggered by the detection system.

References

[1] D. C. Schmidt, V. Kachroo, Y. Krishnamurthy, F. Kuhns, Developing Next-Generation Distributed Applications with AoS-Enabled DPE Middleware, IEEE Communications Magazine, pp. 112–123, October 2000.

[2] D. A. Kaff, C. Rodrigues, Y. Krishnamurthy, I. Pyarali, D. C. Schmidt, Application of the QuO quality-of-service framework to a distributed video application, in proc. of 3rd International Symposium on Distributed Objects and Applications (DOA 2001), october 2001.

[3] L. Romano, N. Mazzocca, A. Coronato, G. De Pietro, Dependability analysis of transmission techniques for MPEG-2 streams, in proc. of Pacific Rim International Symposium on Dependable Computing 2000.

[4] MPEG-2 - International Standard ISO/IEC 13818-1, 1996.

[5] M. Claypool, J. Tanner, The Effect of Jitter on the Perceptual Quality of Video, in proc. of 7th ACM International Conference on Multimedia, October 1999.

[6] W. Jiang, H. Schulzrinne, Perceived Quality of Packet Audio under Bursty Losses, in proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2002), June 2002.

[7] G. Ghinea, J. P. Thomas, QoS Impact on user perception and understanding of multimedia video clips, in proc. of the 6th ACM International Conference on Multimedia, September 1998.

[8] M. Buckley, End to end QoS control in VoIP systems, in the Proc. of Workshop on QoS and User-perceived Transmission Quality in Evolving Networks, Dakar (Senegal), October 2001.

[9] P. Coverdale, Multimedia QoS Requirements from a user perspective, in the proc. of Workshop on QoS and User-perceived Transmission Quality in Evolving Networks, Dakar (Senegal), October 2001.

[10] D. Powell, Failure Mode Assumptions and Assumption Coverage, in proc. of the 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), IEEE-CS, Los Alamitos, CA (USA), 1992, pp. 386–395.

[11] R. Staehli, J. Walpole and D. Maier, Quality of Service Specification for Multimedia Presentations, Multimedia Systems, Vol. 3, nn. 5–6, pp. 251–263, 1995

[12] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, F. Grandoni, Threshold-based mechanisms to discriminate transient from intermittent faults, IEEE Transactions on Computers, Vol. 49, No. 3, pp. 230–245, 2000

[13] L. Romano, S. Chiaradonna, A. Bondavalli, D. Cotroneo, Implementation of Threshold-based Diagnostic Mechanisms for COTS-based Applications, in proc. of 21st IEEE Symposium on Reliable Distributed Systems (SRDS 2002), Osaka, Japan, 2002.

[14] W. Chen, S. Toueg, and M.K. Aguilera, On the quality of service of failure detectors, IEEE Transctions on Computers, 51(5), May 2002, pp. 561-580

[15] M. R. Izquierdo, D. S. Reeves, Survey of Statistical Source Models for Variable-Bit-Rate Compressed Video, Multimedia Systems, Vol. 7, No. 3, pp. 199–213, 1999

[16] P. Wallich, Mindstorms: not just a kid's toy, *IEEE Spectrum*, Vol. 38, N. 9, pp. 52–57, September 2001

[17] D. Laverde, G. Ferrari, J. Stuber, Programming Lego Mindstorms with Java, Syngress Publishing, 2002

[18] P. Verissimo and A. Casimiro, The Timely Computing Base Model and Architecture, in IEEE Transaction on Computers - Special Section on Asynchronous Real-Time Systems, vol. 51, n. 8, Aug 2002

[19] B. Stiller, C. Class, M. Waldvogel, G. Caronni and D. Bauer, A Flexible Middleware for Multimedia Communication: Design, Implementation, and Experience, in IEEE Journal on selected areas in communications, Vol. 17, N. 9, September 1999

[20] J. Kim and K.G. Shin, Performance evaluation of dependable real-time communication with elastic QoS, in Proc. of The International Conference on Dependable Systems and Networks (DSN2001), pp. 295–303

[21] A.M. Hiltunen, R.D. Schlichting, X. Han, M.M. Cardozo and R. Das, Real-Time Dependable Channels: Customizing QoS Attributes for Distributed Systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, N. 6, June 1999, pp. 600–612

[22] D.C. Schmidt et al., Total Quality of Service Provisioning in Middleware and Applications, Elsevier Journal of Microprocessors and Microsystems, Vol. 26, N. 9–10, January 2003

[23] Object Management Group, Control and Management of Audio/Video Streams: Audio/Video Streams v1.0, January 1998

[24] D.C. Schmidt et al., Application of the QuO quality-of-service framework to a distributed video application, in proc. of the 3rd International Symposium on Distributed Objects and Applications (DOA'01), IEEE C-S, 2001, pp. 299–308

[25] A.T. Campbell, G. Coulson, A QOS Adaptive Multimedia Transport System: Design, Implementation and Experiences, Distributed Systems Engineering Journal: Special Issue on Quality of Service, Vol. 4, pp. 48–58, March 1997

[26] Q. Zhang, W. Zhu, Y. Zhang, Resource allocation for multimedia streaming over the Internet, IEEE Transactions on Multimedia, Vol. 3, N. 3, pp. 339–355, September 2001

[27] Y. Xu, D.A. Adjeroh, C.U. Orji, Fault avoidance and recovery for distributed multimedia multicast, in proc. of International Workshop on Multimedia Database Management Systems, IEEE C-S, 1996., pp. 168–174

[28] Sun Microsystems Inc., Java Media Framework API Ver.2.1.1b, 2002
http://java.sun.com/products/java-media/jmf/

[29] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, IETF Request for Comment (RFC1889), 1996
http://www.ietf.org/rfc/rfc1889.txt

[30] ISO/IEC DSM-CC Draft International Standard MPEG-2 Digital Storage Media.Command and Control. ISO/IEC JTC1/SC29/WGll Nll00, November 1995.

[31] XTP-Forum: "Xpress Transport Protocol Specification, XTP Revisions 4.0", 1995

[32] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler and J. Waldo (1999), *The Jini Specification*, Addison-Wesley, Reading

[33] The MathWorks, MATLAB, http://www.mathworks.com

[34] M. Calypool, J. Riedl, End-to-end quality in multimedia applications, Handbook on Multimedia Computing, CRC Press, Boca Raton, FL,1998

[35] S. Varadarajan, H.Q. Ngo, J. Srivarasta, Error spreading: a perception-driven approach to handling error in continuous media streaming, in IEEE/ACM Transactions on Networking, Vol. 10, N. 1, 2002 , pp. 139–152

[36] N. Venkatasubramanian, K. Nahrstedt, An integrated Metric for Video QoS, in proc. of the 5th ACM International Conference on Multimedia, ACM Press, 1997, pp. 371–380

[37] M.P. LaMonte, Y. Xiao, C. Mackenzie, P. Hu, W. Gaasch, J. Cullen, D. Gagliano, TeleBAT: Mobile Telemedicine for the Brain Attack Team, Journal of the American Heart Association, 29:271, 1998.