Towards a Structure-Aware Failure Semantics for Streaming Media Communication Models

Martin Hoffmann, Winfried E. Künnhauser

Computer Science and Automation Faculty, Distributed Systems Division
University of Ilmenau

Abstract

Failure semantics in communication models for distributed systems deal with the impossibility of achieving an exactly-once invocation semantics in failure-prone environments. For remote procedure invocation models, failure semantics such as at-least-once and at-most-once specify guarantees about the number of executions of an invocation as well as its completeness even under the assumption that communication and server failures may occur.

While such failure semantics are quite successful for remote procedure call models they have significant weaknesses when applied to streaming media communication. The main reasons are fundamental differences in the basic communication model as well as case-dependency and granularity of failure treatment in media streams, resulting in awkward abstractions as well as inefficient implementations.

This paper is a step towards an adaptive failure semantics for streaming media communication. We argue that in order to achieve simplicity and economy, failure semantics must dynamically exploit application-level knowledge as well as knowledge from lower system layers, the three corner stones being media stream structure, timing constraints and resource availability.

The paper focuses on structure-awareness as one of these three corner stones. It discusses the role of importance of a media stream fragment, develops a function to quantify importance, and discusses its computability. Experimental results evaluating importance based failure handling conclude the work.

Key words: distributed multimedia systems, communication models, failure semantics, failure service, failure handling policy
1 Introduction

Context of this work are distributed multimedia applications such as traffic control systems, facility and vehicle management, video conferencing, teleteaching and construction assistance systems. These applications typically run on distributed system platforms encompassing a wide range of computing and communication resources, including large multimedia database servers, stationary and mobile workstations, and small hand-held devices.

The fundamental object of communication in distributed multimedia systems is the media stream. Media stream communication differs in many ways from contemporary communication patterns such as client/server request/reply pairs or remote procedure invocations. Media streams are sequential, ordered, hierarchically structured, infinite sequences of discrete data objects that continuously flow from some source to some destinations. In general, media streams are quite voluminous, encompassing compressed video streams such as the MPEG-2 video compression for DVDs with a bit rate of approximately 5 MBit/sec as well as uncompressed HDTV video streams with a rate of 2.8 GBit/sec. Media streams also encompass the notion of time, used to specify periodicity, ordering and timeliness; additionally, time is used to synchronize streams from different sources that belong to the same application\(^1\). Last but not least, communication failures that result in loss or corruption of stream fragments are not always fatal; depending on the stream format and the importance of the affected stream fragment, failures may be tolerated without violating the quality requirements of the application.

Contemporary remote invocation models were not made for such properties. Their domains are bidirectional caller/callee communication patterns with each communication having a well-defined, distinct and relatively small list of parameters. Inevitably, such models can and do not work well for media streams, and several groups are working on more appropriate models and their integration into middleware platforms [11,16,18].

Communication models are composed of several building blocks. As an example, the role model describes what each communicating party is expected to do (e.g. acting as a client or a server), the data model describes properties of the exchanged data (e.g. data defined by signatures in RPC models), and the call and termination semantics describe the control flow properties. This paper focuses on failure semantics that specify guarantees given to the communicating parties even under the assumption that communication failures may occur. Failure semantics simplify failure treatment on the application level by providing easy-to-use models that allow application programmers to

\(^1\) such as audio and video streams in multi-camera and multi-microphone recordings
abstract from low-level details and to specify failure recovery policies on an application-oriented level.

Work on failure handling for multimedia streams focuses for example on adopting stream structure to be more failure resistant [13], conceal errors [5] or be self-corrective [17,2]. Other approaches aim at meeting QoS requirements at the transport layer [12]. In this paper we argue that applicability as well as efficiency of failure handling policies strongly depend on the dynamic failure context, in which application-level knowledge as well as knowledge about current timing restrictions and available CPU, memory, and communication resources play an important role. Such comprehensive knowledge in general is dispersed among several system layers and components and is often neglected by failure services restricted to isolated system layers.

The focus of our work is the influence of application-level knowledge about the structure of a media stream on the efficiency of failure recovery policies. In order to guide the choice between different failure handling policies within a failure service we exploit knowledge about the importance of individual media stream fragments, depending on their role within a stream, and we focus on the importance function as a core element of a structure-aware failure semantics. Chapter 2 discusses briefly the shortcomings of contemporary RPC failure semantics in the context of streaming communication models. Chapter 3 examines the usefulness of a stream fragment’s importance in order to support efficiency and economy of failure semantics, chapter 4 discusses experiments conducted with different fault-injected MPEG-2 video stream types, and chapter 5 summarizes the results.

2 Failure Semantics

Failure semantics in communication models for distributed systems deal with the impossibility of achieving an exactly-once invocation semantics in failure-prone environments. They define easy-to-use models that provide guarantees with respect to a well-defined behavior of communication operations even in the presence of failures. Several examples of failure semantics are available for the remote procedure call [14] communication model. Common RPC models provide a choice between up to four different failure semantics known as exactly-once, at-most-once, at-least-once, and maybe [19,3,10]. While in general there can be no guarantee that the invocation of a remote procedure always results in that procedure being executed exactly once, exactly-once failure semantics for example has a transaction-like behavior. It guarantees that a successful invocation results in the procedure to be executed exactly once or, in case of a failure, the caller is notified, in which case a guarantee is given that the call has not been processed at all. At-most-once semantics
guarantees that in case of a successful return the procedure has been executed
exactly once, whereas in case of a failure the call has either been processed
once, or partially, or not at all. With at-least-once semantics, success of a call
guarantees that the call has been processed at least once. Maybe semantics
finally does not provide any guarantee, reflecting the case when there is no
fault tolerance.

One approach to fault-tolerant multimedia streams is to adopt a well-known
failure semantics that has already proven its virtues. However, applying RPC-
based models to streaming communication models has several disadvantages.

Firstly, procedure calls consider their arguments to be a self-contained and in-
dependent unit. If an entire multimedia stream is considered a single argument
to a single RPC, a communication failure affecting only a small stream frag-
ment would result in the failure of the entire stream communication operation.
On the other hand, dividing a stream into several smaller fragments requires
additional means for maintaining the spatial and temporal relationships be-
tween the fragments, which are no natural property of the RPC paradigm.

Secondly, RPC failure semantics implementation may exploit the bidirectional
RPC call/return paradigm, using the callee’s return message also as an ac-
knowledgement message. In a stream oriented, unidirectional communication
paradigm, a significant overhead due to the need for an additional backward
communication channel would be required.

Thirdly, the best way to react upon a given failure may be different at different
times. Considering real-time applications, a stream fragment that is delayed
beyond its processing deadline may become useless, and an economic failure
recovery policy will abandon such fragment when this occurs. Also, earlier
failure handling may have impacts on subsequent failures. If some stream
fragment was dropped in response to some earlier failure subsequent stream
fragments that depend on the dropped element may become useless, too, and
economy and efficiency require that no resources are wasted for processing
them properly. Last not least, a decision whether to request retransmission of
a corrupted stream fragment should depend on the importance of the fragment
and the availability of CPU, communication, or memory resources.

Concluding, failure semantics for a stream oriented communication model is
influenced by the many-folded relationships between stream fragments, the
unidirectional role model, dynamically changing application requirements, the
stream processing context, and the resources that are available for failure
recovery.

The consequences are two-fold. In order to provide efficient and economic fail-
ure handling, failure semantics must be defined on the fine-grained level of
individual stream fragments, and failure semantics must be capable to dy-
namically adapt to changing circumstances such as timing constraints, past failure handling, and resource availability.

Adapting the RPC model to reflect these properties would put the load of dynamically choosing a failure semantics for each individual stream fragment on the application programmers, burdening them with the monitoring of low-level details such as failure handling histories and resource availability. This clearly violates the fundamental idea behind failure semantics, and we thus will focus on a different approach in which the application merely defines properties and requirements. Taking these, a failure service within the communication subsystem implements the failure semantics by choosing state-dependent adaptive failure recovery policies.

The mechanism to dynamically select such policies constitutes the very heart of an adaptive failure service. The following section focuses on this mechanism and develops a precise foundation for policy selection that generally computes in $O(1)$.

3 Foundations for Policy Selection

Similar to byte sequences representing serialized remote procedure calls, multimedia streams have a well-defined structure; and similar to Sun’s external data representation (EDR) or CORBA’s common data representation (CDR) formats, standards such as MPEG-2 [8] precisely define the format of media streams and its individual elements. Depending on the stream format and the importance of the affected stream fragment, loss or corruption of a stream fragment has a different impact on the stream quality. While failures concerning less important stream fragments (such as B-frames in MPEG-2) might be tolerated without violating the quality requirements of the application, the loss of fragments containing structural information is more liable to have a visible impact on the perceived video quality.

This chapter develops a function for quantifying the importance of a stream fragment based on the stream’s structure. Firstly, in order to prepare for the function’s implementation a simple model for typed streams and typed stream elements is developed that easily maps to contemporary audio and video stream formats. Then, in section 3.2 two basic classes of importance are identified. Both classes then are cast into functions whose composition results in a function that assigns an importance value to a given stream element. Section 3.3 discusses the computability of that function, taking into account the predictability properties of a stream’s element sequence. Finally, because importance of a stream element is not the only dimension that influences a failure recovery policy, the role of the importance function within an adaptive
failure semantics for streaming communication models is discussed.

3.1 Structure Model

Multimedia streams in general contain data from more than one media source. For instance, the DVD version of a motion picture often provides multiple sound tracks together with one or more video tracks. Separate media sources are commonly kept in individual elementary streams, the only relationship between these streams being a common time base. Elementary streams in general have a logical structure depending on the stream type (consisting for example of clips, scenes, frames and slices in many video stream type specifications). Stream fragments that correspond to a logical structure element (such as a frame or a slice) will be denoted as stream objects throughout this paper. In order to simplify transmission elementary streams are often multiplexed into a single transport stream exhibiting a much simpler packet structure. In general, a packet may contain several stream objects (which in practice belong to the same elementary stream), and a stream object – while passed through a communication system – may be distributed among more than one packet.

In general, a stream model would thus have to distinguish between elements of elementary streams (stream objects such as frames and slices) and elements of transport streams (packets). However, the transport stream’s packets contain parts of objects. If a failure occurs at the transport level, thereby corrupting packets, it also affects parts of the object that is stored within those packets. Thus a failure occurring at the transport level translates into one at the elementary stream level. Therefore, in order to keep our stream model simple, the model will only consider elementary streams\(^2\) and their objects.

A basic property of all streams is that its objects are ordered. Any observer will see one stream fragment after another. This is especially true for multimedia streams where order is an inherent property of the data. We therefore describe a stream  \(S\) as a sequence of stream objects from some basic set  \(O\):

\[
S = [s_i], \quad i \in \mathbb{N}^0, s_i \in O.
\]

Current standards for multimedia streams such as MPEG define objects as instances of types [9]. In order to obtain the type of a stream object, we will use a function

\[
otype : O \rightarrow T
\]

\(^2\) We will use the term "stream" instead of "elementary stream" as long as its meaning is unambiguous.
with $T$ being the set of all known object types. However, because of the sequential transmission of a stream an observer may not be able to discover the type of an object until it has been seen. The border between the objects seen by an observer and those still invisible will be called the observer’s horizon. If the type of an object is determined from the object itself, then for any observer the function $otype$ is only defined for those objects that are before its horizon. However, depending on the structure definition of a stream the type of objects beyond an observer’s horizon may be predicted; we will look into predictable streams in section 3.3.

### 3.2 Importance

If a decision is to be made about the fate of an object in case of a failure the importance of the object needs to be accounted for. Importance of an object depends on the information encoded within the object as well as the temporal and spatial terrain that is lost if the information is lost. Because codecs in general remove redundancy, information within one object is often reused by following objects. Therefore, the information encoded in an object may be more persistent than the object itself. As a consequence, our model distinguishes between two classes of information: the core information of an object that is relevant only for the object itself and meta information that is also used by following objects.

Both classes need to be rated separately for their importance. The core importance of an object depends on the core information of an object. Its importance reflects the temporal and spatial terrain it covers and is determined by an object’s type as defined in a stream specification. Within our model, the function $core$ assigns core importances to object types.

$$
core : T \to \mathbb{R}.
$$

Meta importance of an object relates to the core importance of all objects that depend on that object. We thus define it to be the sum of the core importance of all dependent objects. Because number and type of dependent objects may be different for different objects of the same type, meta importance needs to be determined for every object individually. Formally, we define the function $meta$ to assign meta importance to an object:

$$
meta : O \to \mathbb{R}.
$$

The overall importance $imp : O \to \mathbb{R}$ of an object consists of both meta and
core importance of an object. Modelled as a function we obtain

\[ \text{imp}(s_i) = \text{core}(\text{octype}(s_i)) + \text{meta}(s_i), \quad s_i \in O. \]

Overall importance can be used to determine meta importance recursively. Instead of collecting all dependent objects and summarizing their core importance we can collect directly dependent objects only and sum up their overall importances.

### 3.3 Predictability

In order to compute the actual importance of a stream object the value of \textit{meta} as one of the two components has to be known. For this, knowledge about all objects that depend on the information in a given object is needed. However, as long as a stream is processed strictly sequentially all dependent objects are still beyond the observer’s horizon. Thus it becomes necessary to gather information about objects behind the horizon or, in other words, we need to predict the future of a stream.

This is possible to a certain degree only and depends on the rules of the make-up of the stream. Since those rules are defined in terms of object types the degree of predictability can be defined for object types as well. This degree, however, is only relevant to predicting the number and type of directly dependent objects. Predicting their dependent objects may be governed by other propositions. This phenomenon is related to the recursive definition of meta importance above. If we restrict predictability to directly dependent objects we may identify three different predictability classes.

Firstly, for some object types the number and type of directly depending objects are always the same. Such object types are called \textit{strictly predictable}. For a second class of object types the number and type of dependent objects are known only when preceding parts of the stream are successfully processed; these objects are called \textit{limitedly predictable}. Finally, all object types which are neither strictly predictable nor limitedly predictable are called \textit{unpredictable}.

Consequently, meta importance can only be determined exactly if all object types of the stream are strictly predictable. A single object type within the stream that is not strictly predictable already renders meta importance to be no longer computable. However, as we will discuss in the next section, heuristic approaches based on the specification of different stream types (such as MPEG-2) will allow for an appropriate approximation.

The consequence is that whether meta importance can be calculated exactly or heuristically depends on the particular stream type. The function \textit{meta} thus
is stream-type dependent. While being a constant value for one stream type, it may (for streams that are not strictly predictable) become a more complex expression for the next.

3.4 Composition of an Evaluation Algorithm

Adaptive failure semantics encompass a variety of reactive and proactive failure recovery policies. Whenever a failure occurs, one such policy is dynamically selected for failure handling, the choice depending on each individual failure context.

Policy selection based on individual failure contexts has many dimensions. Reactive failure recovery policies for example that are based on retransmission of a lost or corrupted stream fragment have several conditions that must be met. Firstly, there are resource-related necessary conditions such as the sender maintaining a copy of the fragment, communication bandwidth availability, and the availability of sufficient time before the fragment’s processing deadline. Then, there are sufficient conditions depending on the application level context, reflected by current QoS contracts and the importance of the stream fragment affected by the failure, the latter depending on the fragment’s temporal and spatial sphere of influence. Moreover, proactive failure avoidance policies triggered by failure amassment additionally require histories about failures, resource consumption and application behavior.

The implementation of a failure semantics (referred to as the failure service) is part of the communication model’s implementation generally located within middleware layers or distributed and network operating systems.

Whenever a failure occurs, a failure service executes three steps. Firstly, in order to select a failure recovery policy, information about the current application-level, resource and time contexts are gathered. This is the role of the evaluation this section is about. Secondly, the failure service decides upon an adequate recovery policy, and, as the last step, executes this policy.

Importance of a stream fragment as modelled by the importance function $imp : O \rightarrow \mathbb{R}$ in section 3.2 has a major impact on policy selection. Even in real-time multimedia applications where stream communication is subject to strict timing constraints, a failure recovery policy based on retransmission that does not meet a fragment’s deadline may still be important to prevent further error propagation [17].

However, importance is not the only dimension that influences the choice of a failure recovery policy. In video stream formats without error propagation (such as the DV digital video format (IEC 61834)), meeting deadlines becomes
a necessary condition.

Another dimension to be considered is the extensive resource requirements of multimedia applications. Because the amount of resources available to an application usually is limited, failure recovery competes with regular stream processing. In order to meet overall quality requirements resources assigned to failure recovery must be well balanced with resources assigned to regular stream processing.

All three dimensions – object importance, timing constraints, and computational resources – are important issues for the selection of a failure recovery policy. However, because this paper’s focus is the importance dimension alone, timing constraints and computational resources are represented by two abstract boolean functions $t : A \rightarrow \{true, false\}$ and $r : A \rightarrow \{true, false\}$ that model the necessary timing and resource conditions for some policy from the policy set $A$. In other words, these surrogate functions describe the necessary conditions to be met for the success of a chosen policy, while being selected by $imp$ is the sufficient condition. This approach allows us to focus our model as well as its implementation on the importance function alone, and will also provide a plug-in for future work on timing and resource conditions.

Concluding, the variety of failure recovery policies within an adaptive failure semantics requires that choice is made whenever a failure occurs. We propose that such a choice takes into account three dimensions: the importance of a stream fragment (application level context), the timing context and the resource context. These dimensions are modelled by three functions $imp$, $t$ and $r$, reflecting necessary and sufficient conditions for a successful policy execution.

4 Experimental Model Evaluation – A Case Study

Intuitively, there is a difference whether the policy to handle a faulty stream fragment is either the same for every fragment or whether knowledge about the importance of the involved stream objects within the faulty fragment comes to bear. In order to support this intuition we applied our approach experimentally to several different examples of fault-injected MPEG-2 video streams. In order to obtain results that are not obscured by physical influences, and that focus on the effects of importance variations alone we preferred a simulation model to a real world measurement environment.
4.1 Scenario

Failure services in distributed multimedia applications can be used for more than a single purpose. The obvious one is to implement policies for dealing with communication failures, causing loss or corruption of stream objects. Here a faulty object is the input for the evaluation function which decides how to react to the failure. However, it is rather difficult to simulate this scenario accurately and at the same time focus only on the importance of objects. The reason is that other dimensions such as timing and resource constraints play an important role, too. A realistic simulation would require to include these issues in the model, thus obscuring those effects of the importance function we are interested in.

We simulated a slightly different scenario in which the importance function is applied to select stream fragments that have to be dropped as the result of limited resources. The typical example is a buffer queue overflow due to insufficient memory, communication, or processing resources. As a response to a buffer allocation failure, some fragment of a stream has to be dropped, and we are interested in the increase of quality obtained by taking the importance of a stream fragment into account.

This scenario translates into a simulation model where a stream of a certain size has to be shrunk to a smaller size. The amount of shrinking is described by the drop rate \( d \),

\[
d = 1 - \frac{\text{departure rate}}{\text{arrival rate}}.\]

Figure 1 illustrates the simulation scenario.

\[\text{Fig. 1. The simulation scenario}\]

4.2 The Reference Case

If arrival and departure rates of a queue are assumed to be fairly constant and the arrival rate is higher than the departure rate, then there is a probability greater than zero that a specific stream element has to be dropped, and this probability is almost constant for the entire stream. If we do not take stream
structure into account by considering the importance of individual objects, this can be modelled by dropping objects with a fixed probability. We will refer to this case of equally distributed drop probabilities as the *reference case*.

Because a real world drop policy as simulated in the reference case is in general not aware of a stream’s structure it will not be able to identify individual objects (as we assume in our simulation). However, because it is well known that fragmenting a stream without considering its structure results in bad communication error recovery, splitting the stream into appropriate stream objects is usually done by the encoding software. Thus it actually does reflect reality if we use stream objects as fragments even if the failure service is actually not aware of a stream’s structure.

Consequently, simulating the reference case by dropping every object of the stream with the same probability equal to the predefined target drop rate is rather close to reality. We model this by assigning the same fixed drop probability \( p(t) \) to every object type:

\[
\forall t \in T : \ p(t) = d
\]

4.3 The Structure-aware Case

While having the same overall drop rate as the reference case the structure-aware case drops stream objects according to their importance. For this we first have to identify the objects of MPEG-2 video streams, then we have to determine their importance and finally we will use this importance to calculate drop probabilities for every individual object.

4.3.1 Stream Objects in MPEG-2 Streams

The specification of MPEG-2 video streams uses bit patterns that uniquely identify the start of structural elements such as sequence headers, picture headers, slices, or extensions. Thus the type of objects from slice upwards can be determined by parsing the stream for said bit patterns. These structural elements are MPEG-2 *stream objects* as introduced in section 3.1.

4.3.2 Design of an Importance Function \( \text{imp}^* \) for MPEG-2 Streams

In MPEG-2 video streams all core information is contained in slice objects, and thus only slices have a core importance. Some MPEG-2 profiles (precisely described applications of the rather loose MPEG-2 specification) require that
each picture is made up from the same number of slices of the same size. If we use such a profile, all slices cover the same spatial area and are visible for the same amount of time. They therefore have the same core importance, which for convenience will be modelled by the numerical value 1.

Slices may carry meta information, too, because they may be used in motion compensation of other slices. How many slices depend on a given slice is determined by the picture class a slice belongs to. MPEG-2 video streams are made up from three different picture classes, namely \textit{I} (\textit{intra}) pictures that are fully self-contained, \textit{P} (\textit{predicted}) pictures whose slices may depend on slices of the preceding I or P picture, and \textit{B} (\textit{bidirectional}) pictures whose slices may depend on slices of the two preceding\footnote{with respect to their transport order} I or P pictures.

Because I pictures have no references to prior pictures all picture sequences beginning with an I picture and ending with the last picture before the next I picture constitute a self-contained group. The length of such a group can vary and is generally not known beforehand. Therefore, MPEG is an example of an unpredictable stream, and assumptions have to be made in order to estimate meta importance. We assume that each group is 12 pictures long and is made up in the way shown in figure 2. This is a default for European Television video, because it covers half a second of European Television’s 24 pictures per second. Most encoders use this default when there is no need for groups of a specific length.

![Fig. 2. The structure of an MPEG group in 24 pictures/second video stream](image)

In order to determine the meta importance of a given slice its picture class has to be known, making it impossible to assign a static meta importance to slice objects. This can be achieved, however, by introducing different object types for slices of the different picture classes. Meta importance and thus overall importance then depends on the object type only. We express this by a slightly modified importance function $\text{imp}^* : T \rightarrow \mathbb{R}$. $\text{imp}$ is starred for reasons of formal correctness only: unlike the original $\text{imp}$ function from section 3.2 which maps objects to importance values, $\text{imp}^*$ operates on object types.

Table 1 assigns numerical values to $\text{imp}^*$ that relate to the importance of the different object types in MPEG-2 video streams. The rational behind the values in column 2 is that they count the number of objects that depend on any object of the respective type, thus reflecting its spatial and also temporal impact. Thus any I-slice has an importance value of 14, because all 12 objects of its group depend on it, plus 2 B-slices from the previous group. In the same
way, 11 frames depend on the first P-frame (frames 1-11 in figure 3), while only 1 frame depends on any B-frame slice (the frame the slice belongs to).

Note that \textit{imp} operates on object types only and does not take into account the position of an object within a stream. It thus does not catch the slight difference that occurs whenever a new sequence starts. The last two B pictures of a group depend on the I picture of the following group only if both groups belong to the same sequence. Therefore, slices of the first I picture in a new sequence have only 12 objects dependent on it. However, the simplified approach to generally assign an importance value of 14 to any I slice allows for a stateless and faster implementation of \textit{imp}. Any negative effect of this simplification only appears in borderline cases where drop rates are extremely high and where even I slices are dropped. Because in that case the PSNR value is already well below 20dB (see section 4.4), we consider the gained computing speedup to justify this simplification.

In a similar way, layered video coding techniques supporting graceful quality degrading (such as \textit{quality assurance layering} [15,7] or \textit{layered DCT} [2]) also provide a straightforward (yet coarse-grained) importance heuristics based on the essential and enhancement layers. Additionally, coding techniques such as [6,1] provide importance refinement by introducing a temporal hierarchy within each layer, based on the MPEG I/P/B picture pattern.

Objects other than slices only have meta importance. However, they are very important, indeed. With regard to the time span their data is used, the least important is the picture header which already affects an entire picture and – due to motion compensation – maybe even parts of other pictures. In other words, these objects are important enough to be never dropped at all. This is reflected in the \textit{imp} implementation by assigning the maximum importance value to such objects.

4.3.3 Drop Probability

Now that we know the importances of all objects involved we have to turn them into drop probabilities. Objects containing meta information (such as scene objects) have maximum importance and in fact are never dropped at all; they are assigned a drop probability zero. For slices we have to find a mapping that (a) represents their importance (or, more precisely, the difference between their respective importances) and (b) sums up to the predefined percentage of the stream to be dropped.

(b) requires that the drop probability is put into relation to the frequency a slice type appears within the stream. For our example scenario with a group length of 12, table 1 contains this frequency in the second column. The sum of the drop probabilities \( p(t) \) of the various types weighed by their respective
Next we have to find values for \( p(t) \) that reflect the importance of the object types as shown in Table 1. First of all, the drop probability is inversely proportional to the object’s importance. But since using just \( \frac{1}{imp^*(t)} \) is unlikely to satisfy equation (1) we scale this expression by a yet to determine scale factor \( a \) and obtain

\[
p(t) = a \frac{1}{imp^*(t)}.
\]

(2)

By inserting (2) into (1) we find that \( a \) depends on \( d \) such that

\[
a = d \left( \sum_{t \in T} \frac{1}{imp^*(t)} f(t) \right)^{-1}.
\]

(3)

A problem here is that \( p(t) \) is not yet a probability value between 0 and 1. By just scaling \( \frac{1}{imp^*(t)} \) with some \( a \), \( p(t) \) may grow larger than 1 if \( \frac{1}{imp^*(t)} \) for some object type is very large compared to the values of the other object types. In this case dropping all objects of this type will not be sufficient to achieve the given drop rate. The solution is to increase probabilities for the other types. This can easily be done by increasing \( d \).

While applying equations 1 to 3 already leads to acceptable experimental results, they still can be improved. Scaling the respective importances \( p(t) \) linearly keeps the drop probabilities rather close together. If instead of a linear factor \( a \) we apply an exponential function that emphasizes drop probability differences, more B picture slices will be dropped instead of slices from I and P pictures. A good practical choice for doing so is the function \( e^x \); it returns
small values for small inputs but quickly increases to very large values for growing inputs, thus increasing their differences\textsuperscript{4}. Additionally, we will not use the pure $e^x$ function but scale it with a factor $b$ in such a way that the difference between smallest and largest importances is maximized. We thus substitute the term $imp^*(t)$ in equations (2) and (3) by the term $e^{b \cdot imp^*(t)}$.

To determine the scaling factor $b$ we have to maximize a function $g(b)$ with

$$g(b) = \left(e^{b \cdot imp^*(B\text{-slice})}\right)^{-1} - \left(e^{b \cdot imp^*(I\text{-slice})}\right)^{-1}$$

$$= e^{-b} - e^{-14b}.$$ 

This function has its maximum at $b = 1/13 \cdot \ln 14$.

In conclusion, the drop probabilities for the respective object types in the structure-aware case are determined by

$$p(t) = a \cdot e^{-b \cdot imp^*(t)}$$

with the constant factor $a$ such that

$$a = d \left(\sum_{t \in T} e^{-b \cdot imp^*(t)} \cdot f(t)\right)^{-1}$$

and $b$ as outlined above.

4.4 Measuring Quality

The idea of measuring quality usually is to reduce all the appropriate parameters to a single quality value. With MPEG video the peak signal-to-noise ratio (PSNR) is often used. For this, the mean-squared error $\sigma^2$ is required. If we have recorded $N$ sample values, $\sigma^2$ is calculated by

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (o_i - r_i)^2,$$

where $o_i$ represents a sample and $r_i$ the corresponding reference signal.

\textsuperscript{4} Note that we can choose any function to scale $imp^*(t)$ here without raising accusations to manipulate the results because we are merely calibrating the importance in such a way that it has the largest impact.
In order to provide a measure that is independent of the maximum intensity, this value is then scaled by the maximum intensity (in our samples 8 Bits, \(2^8 - 1 = 255\)) of the signal:

\[
PSNR = 10 \log_{10} \frac{255^2}{\sigma^2}
\]

The logarithm has been introduced because the values are rapidly growing. The result is that the PSNR is always given in decibels. As an example, the MPEG-2 video compression for DVDs with a bit rate of approximately 5MBit/sec typically achieves results within the range of 20dB to 40dB. Figure 3 shows a sample picture from a stream processed with various resulting PSNR values. It shows that pictures with a PSNR of 13dB have almost no resemblance with the original while 20dB may be acceptable for small periods of time in certain applications.

![Sample Picture with Various PSNR Values](image)

\(\infty\) dB

20 dB

15 dB

13 dB

Fig. 3. A sample picture from an MPEG stream with different PSNR values

PSNR has often been criticized for being inappropriate for measuring the quality of video (see, for instance, [4]). Arguments are that the quality should be measured according to how viewers perceive it. The metrics should therefore be based on the physiology and psychology of the human visual system. Nonetheless, PSNR is widely used because it is a very simple method and
there are currently no other widely accepted models. However, we should keep in mind that the quality measured represents the quality of the signal determined by decoding the stream and not the quality as perceived by viewers, and we can only interpret the results of the experiments as technical values.

4.5 The Experiments and Their Results

Removing redundancy leads to quite different MPEG streams for different video material. While a video conference scenario for example is mainly motionless, a pop music video contains many fast movements and cuts, and one might expect that experiments with such different video material also have different results. In order to cover a broad test area, we chose video sequences from four different scenarios:

- **Video conferencing.** Streams have long scenes with almost no motion and virtually no cuts. The test sequence is a 720 x 576 frame size, 25 frames per second, 7.5Mbits/second sequence of 737 frames with a fixed GOP size of 15.

- **Pop music video.** Streams are full of movement in many layers with almost no constant background and have many cuts. The test sequence is a 720 x 576 frame size, 25 frames/second, 7.5Mbits/second sequence of 768 frames with a variable GOP size between 9 and 15.

- **Soccer.** The sequence is full of movement but has a very steady background with only few cuts. The test sequence is a 720 x 576 frame size, 25 frames/second, 7.5Mbits/second sequence of 826 frames with a variable GOP size between 9 and 15.

- **Susi.** The sequence is one of the standard clips frequently used in the MPEG video codec research community. The test sequence is a 352 x 288 frame size, 25 frames/second, 1.5Mbits/second sequence of 374 frames.

Figure 4 compares the PSNR values of MPEG-2 video streams resulting from dropping objects according to their importance with an equally distributed dropping probability for these four scenarios.

The charts show a significantly better quality if objects are dropped with an awareness of structure. The PSNR values are up to 15dB better, with an average of well over 10dB. Furthermore, the PSNR always remains above the critical 20dB value. In terms of tolerated frame losses and with respect to the 20dB quality limit, importance-aware frame dropping tolerates a more than 5 times higher frame loss rate in the higher-resolution test applications. This strongly indicates that object importance has a major influence on the efficiency of the evaluation function which in turn is one of the three cornerstones of an economic failure recovery policy.
5 Summary

The paper is a step towards an adaptive failure semantics for streaming media communication models. It discusses structure-awareness within a failure service as one of the three corner stones for dynamically selecting failure handling policies.

The idea is based on the observation that media stream formats are well-defined, that different stream elements generally are of different importance to an application, and that a computable function exists that dynamically assigns importance values to stream elements. Using this function, a failure service is in the position to dynamically adapt its failure handling policies to the importance of lost or corrupted stream elements.

In order to validate the approach, experiments have been conducted comparing arbitrary and importance based dropping rates. A stream-specific importance function for MPEG-2 video streams was developed and applied to four different video stream samples. The results show that our approach maintains an average PSNR improvement between 6dB and 20dB even under heavy packet loss, and that frequently an at least 5 times higher frame loss rate is tolerated at the 20dB quality level. Two other important dimensions of an adaptive failure semantics – timing and resource constraints – are subject to ongoing work.
References


