Analysis of a Multimedia Stream using Stochastic Process Algebra*

H. Bowman, J.W. Bryans and J. Derrick

Computing Lab., U. of Kent, Canterbury, Kent, CT2 7NF, UK
Tel: +44-1227-827570 and Fax: +44-1227-762811
{H.Bowman,J.W.Bryans,J.Derrick}@ukc.ac.uk

Abstract

It is now well recognised that the next generation of distributed systems will be distributed *multimedia* systems. Central to multimedia systems is quality of service, which defines the non-functional requirements on the system. In this paper we investigate how stochastic process algebra can be used in order to determine the quality of service properties of distributed multimedia systems. We use a simple multimedia stream as our basic example. We describe it in the Stochastic Process Algebra PEPA and then we analyse whether the stream satisfies a set of quality of service parameters: throughput, end-to-end latency, jitter and error rates.

1 Introduction

It is now well recognised that the next generation of distributed systems will be distributed *multimedia* systems, supporting multimedia applications such as video conferencing. Importantly though, multimedia imposes a number of new requirements on distributed computing, not least of which is the need to ensure "timely" transmission and presentation of multimedia data, e.g. ensuring that the end-to-end timing delay between transmitting and presenting video frames stays within acceptable bounds. Such real-time constraints are typically embraced by the concept of *quality of service* [BBBC98].

Quality of Service (QoS) characterizes the *non-functional* properties of a system; it is expressed in terms of a number of quantifiable criteria, e.g. timeliness, capacity, integrity, cost, security, reliability and priority. In this paper we focus on real-time QoS parameters, such as throughput, end-to-end latency and jitter, we will clarify these concepts shortly.

Traditionally, in the field of real-time systems, fulfilment of real-time requirements is ensured by a process of measurement and refinement. However, such approaches are usually informal and there are examples of finished systems which are rendered worthless because they cannot meet their real-time requirements. In the field of distributed systems, the role of ensuring real-time requirements are met falls on QoS management [HCCB94]. Attempts can be made to provide the required quality of service through a combination of QoS management functions including resource reservation and admission control, monitoring and adaptation. Again, however, such measures are undertaken after the system is deployed.

It is also worth noting that QoS management is a notoriously difficult activity. Specifically, QoS capabilities change dramatically as the load on a system varies; such contention for bandwidth implies that QoS is a highly dynamic measure and is difficult to determine statically. Furthermore, QoS is fundamentally an end-to-end measure; localized measurement is only a partial solution. In addition, end-to-end measurement must typically be made in a highly heterogeneous setting, across administrative and management domains [Slo94].

[0]The research presented here is supported by the UK Engineering and Physical Sciences Research Council under grant number GR/L95878 (A Specification Architecture for the Validation of Real-time and Stochastic Quality of Service).

It is clear that attempting to quantify the performance of a system once it is built will not always yield a reliable measure of QoS capabilities. Information on performance capabilities need to be determined during system development and be used to inform dynamic measurement systems.

In response, a number of researchers have considered techniques for the specification [BBBC98, FL98] and verification [BFM98] of Quality of Service. However to date, this work has been restricted to specification and verification using *deterministic timing*, e.g. putting fixed upper and lower bounds on the time that actions are offered to the environment. This is a useful first step, but it does not lead to a very refined model of the performance of systems. It is also necessary to consider *probabilistic* and *stochastic* concerns, for example to reason about the distribution of timings on packet deliveries or the probabilities of packet loss.

This paper makes a first step in this direction by assessing the suitability of stochastic process algebras for the specification and analysis of distributed multimedia systems. Stochastic process algebras are now a relatively extensively investigated topic, with a number of techniques and tools available, e.g. PEPA [Hil96], TIPP [HRW95], EMPA [BDG95] PA_{GS} [Kat96] and SPADES [DKB97]. Here we consider one of the most important techniques, PEPA. Our approach is to model an existing example of a multimedia system, a multimedia stream, in PEPA and then investigate how to check that the system satisfies certain real-time quality of service properties.

The work being reported here has been performed in the context of the V-QoS project which is an EPSRC funded project between the University of Kent at Canterbury and Lancaster University.

Structure of paper. First we give background on distributed multimedia systems in Section 2, and in particular, we introduce the multimedia stream example. Then in Section 3 we review the stochastic process algebra PEPA. In Section 4 we give a PEPA specification and analysis of the multimedia stream. In Section 5 we discuss the use of immediate actions in stochastic process algebra. Then in Section 6 we assess the suitability of PEPA for such specification and analysis in the light of Section 4 and we give pointers to further work.

2 Distributed Multimedia Systems

2.1 Background

It is typically argued that the incorporation of multimedia enforces three new requirements on distributed systems [BBBC98]:-

- *Continuous Interaction.* Traditionally, distributed systems communication paradigms support interaction of a logically singular character, e.g. a remote procedure call. However, the advent of multimedia means that this is not sufficient. In particular, interaction of an "ongoing" nature must be provided, e.g. continuous transmission of video frames in a video conferencing application. Such an ongoing interaction is called a *stream* (the term *flow* is also often used). We call the elements that are transmitted in a stream *packets*.

- *Quality of Service.* QoS requirements also have to be associated with such continuous interactions. For example, if in a video conferencing application, the end-to-end delay between the generation of frames and their presentation becomes too great the sense of simultaneous interaction will be lost. Typical quality of service properties include: *end-to-end latency* (delay) between the generation of packets and their presentation, *throughput*, i.e. the rate at which packets are presented and *jitter*, which is a measure of the variability of delay [BBBC98]. Limiting jitter ensures that there is not an unacceptable variability around the optimum presentation time, e.g. if one packet is presented quite early and the next is presented relatively late an unacceptable stutter in the presentation may result.

- *Real-time Synchronisation.* It is also often necessary to synchronise multiple media streams. For example, in order to enforce lip-synchronisation, video and audio streams must be synchronised.

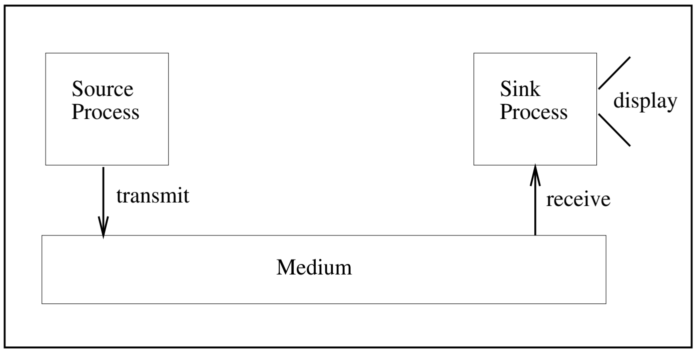

Figure 1: A Multimedia Stream

Application specific real-time synchronisation also arises, e.g. if captions need to be displayed at particular points in a video presentation.

The simple multimedia stream, which we present next, illustrates the first two of these requirements. Unfortunately, it is beyond the scope of this paper to consider real-time synchronisation, however, we can point the interested reader to a number of papers which specify a lip synchronisation algorithm using process algebras, e.g. [Reg93, BBBC98, ABSS96, BFM98].

2.2 The Multimedia Stream

The basic multimedia stream is as depicted in Figure 1. It has three top level components: a *Source* process, a *Sink* process and a communication *Medium*. The *Source* generates a continuous sequence of packets[1] which are relayed by the *Medium* to the *Sink*, which then displays them. The *Medium* is assumed to support asynchronous communication between the *Source* and the *Sink*. In addition, the *Medium* is unreliable and may lose messages. Three basic actions support the flow of data (see Figure 1 again), *transmit*, *receive* and *display*, which respectively signal the transfer of packets from the *Source* to the *Medium*, from the *Medium* to the *Sink* and their display at the *Sink*. In our stochastic analysis, specific rates will be assoicated with the actions *transmit*, *receive* and *display*.

This example is based upon the LOTOS/QTL specification that appears in [BBBC93, Bla94, BBBC98]. However, the formulation of the stream in [Bla94] contains specific timing assumptions, e.g. that the *Sink* takes 5ms to process frames and error behaviour, e.g. that if a frame arrives particularly late then the system should go into an error state. A theme of the sequel is to see to what extent we can reflect these timing assumptions in the setting of a PEPA analysis.

In Section 4, we present a PEPA description of the basic stream behaviour and focus on our main objective: to analyse the quality of service properties of the stream. We will vary parameters in the system and see what consequences they have on a number of quality of service properties. The QoS properties we will consider will be, *latency*, the end-to-end delay between a *transmit* action and its corresponding *display* action; *throughput*, the rate at which the *Sink* process *display*s packets; *jitter*, which quantifies how latency values vary about the optimum; and the *error rates* at which the system can go into error.

[1]These could be video frames, sound samples or any other item in a continuous media transmission. In this way the scenario remains completely generic. However, instantiation of data values specializes the scenario.

3 The Stochastic Process Algebra PEPA

Process algebras are a mature formalism for describing and analysing concurrent and distributed systems; important process algebra approaches include CCS [Mil89], CSP [Hoa85] and LOTOS [BB88]. Furthermore, there are now a number of approaches for incorporating stochastic features into process algebra, e.g. [Hil96, HRW95, BDG95, Kat96, DKB97]. It is argued [Hil96] that stochastic process algebras offer a number of benefits over standard performance analysis techniques such as queueing models [Kin90] and Petri Nets [MBC+95], not least of which is that stochastic process algebra enable *compositional* description of performance issues.

The particular stochastic process algebra we consider is PEPA [Hil96]. Within PEPA, every *activity* (so called to distinguish it from process-algebraic actions) has a duration. However, an *event* — what the observer sees when an activity finishes — is instantaneous. An activity a is defined as a pair (α, r) where $\alpha \in \mathcal{A}$ is the *action type* and r is the *activity rate*. Each activity is uniquelly typed. τ is the unknown type.

The duration of each PEPA activity is determined by an associated *exponential* probability distribution function. This function is parameterised by the *activity rate*, which is either a real number or $\top$ — the *unspecified* rate. When enabled, the activity $a = (\alpha, r)$ will delay for a period determined by its distribution function: the probability that a happens within time t is given by $F_a(t) = 1 - e^{-rt}$.

The syntax of PEPA is given by

$$P := (\alpha, r).P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \mid A$$

where P is a process, L is a set of actions and A is a constant. We assume a countable set of process definitions $A \stackrel{def}{=} P$. These terms represent, prefix, choice, coopoeration, hiding and process instantiation. For definitions of these operators the reader is refered to [Hil96].

Having specified a system in PEPA, it can be analysed using the PEPA Workbench [Gil97]. Any finite PEPA process has an underlying Markov chain; this fact forms the basis of all the analysis that is performed. The PEPA workbench generates this Markov chain which can then be solved to determine the underlying probability vector. This vector characterises the equilibrium behaviour of the PEPA specificaton: elements of the vector give the (steady state) probability that the specification is in a particular state. As illustrated later, a number of performance measures can be derived from these steady state probabilities.

4 PEPA Specification of the Stream

4.1 Specification

We model the stream as a composition of four components: a *Source*, a *Channel*, a *Sink* and a *Timer*. The complete specification is given by

$$Source \underset{\{transmit\}}{\bowtie} Channel \underset{\{receive\}}{\bowtie} Sink \underset{\{reset\}}{\bowtie} Timer$$

We describe each component in turn.

Source. The *Source* simply transmits frames onto the medium at a rate of r_{trans}; we specify it as,

$$Source \stackrel{def}{=} (transmit, r_{trans}).Source$$

Channel. The *Channel* component models the medium; it accepts frames from the source (via the action type *transmit*) and then either passes them on to the *Sink*, (via the action type *receive*, with rate r_{rec}), or loses them (via the action type *loss*, with rate r_{loss}). A perfect channel may be described by setting r_{loss} to zero.

We model the *Channel* as a finite buffer holding up to five frames[2]. The complete description is as follows. Although not strictly allowed by the PEPA syntax, we parameterise the definition of *Channel* in order to simplify our presentation.

$$\begin{array}{rcll}
Channel & \stackrel{def}{=} & Channel_0 & \\
Channel_0 & \stackrel{def}{=} & (transmit, \top).Channel_1 & \\
Channel_i & \stackrel{def}{=} & (transmit, \top).Channel_{i+1} + & 1 \leqslant i \leqslant 4 \\
 & & (receive, r_{rec}).Channel_{i-1} + & \\
 & & (loss, r_{loss}).Channel_{i-1} & \\
Channel_5 & \stackrel{def}{=} & (receive, r_{rec}).Channel_4 + (loss, r_{loss}).Channel_4 &
\end{array}$$

The *transmit* action type in *Channel* is passive (the medium can accept frames from the *Source* at any rate). In fact, to use the Workbench to analyse the specification, *transmit* must be passive since the current version of the PEPA Workbench requires that only one action type instance may influence the corresponding activity rate.

In the untimed setting the action *loss* would be hidden from the environment, we could use the PEPA hiding operator to obtain the same effect with PEPA. However, in contrast to in the (deterministic) timed case, where hiding enforces maximal progress [Reg93], here it does not effect the results of Markov analysis, thus, we do not include it.

Sink. The *Sink* (modelled as a three place buffer) receives frames and displays them. The *receive* action type is passive (any rate of frames is accepted).

$$\begin{array}{rcll}
Sink & \stackrel{def}{=} & Sink_0 & \\
Sink_0 & \stackrel{def}{=} & (receive, \top).Sink_1 & \\
Sink_i & \stackrel{def}{=} & (receive, \top).Sink_{i+1} + (display, r_{disp}).Sink_{R(i-1)} & 1 \leqslant i \leqslant 2 \\
Sink_3 & \stackrel{def}{=} & (display, r_{disp}).Sink_{R2} & \\
Sink_{Ri} & \stackrel{def}{=} & (reset, r_{reset}).Sink_i & 0 \leqslant i \leqslant 2
\end{array}$$

Error Rates. In the deterministic case, an error is typically signalled by forcing the system to enter an error state (which would typically be a stop state) when certain behavioural properties are invalidated, e.g. the level of throughput goes out of certain bounds [BFM98]. However, this is not possible within the PEPA formalism since in order for Markov analysis to be performed, the specification must be irreducible [Hil96]. The existence of a deadlock state would invalidate irreducibility. Consequently, in this paper we investigate an alternative form of error behaviour. The approach is that if the gap between consecutive *display*s is beyond a certain threshold level, then the system simply signals an error, by performing an *error*. Such signals could be used in a network management backbone where error rate statistics are accummulated.

In order to model this error behaviour we use a *Timer* component. The job of *Timer* is to monitor the delay between *display*s, and to report an error if the delay exceeds a certain limit. After each *display*, the *Sink* sends a *reset* to the *Timer*. The *reset*s are signals to the *Timer* (which synchronises on them), and we would naturally like to model them as immediate actions. Although some attempts have been made to allow instantaneous actions within stochastic process algebras (see for example [HRW95]), they are not included within PEPA. We therefore model signal activities by setting the rate to be much greater (by a factor of 10 in our example) than the rate of any of the other activities. We will return to the issue of immediate actions in Section 5.

[2] We cannot model an infinite buffer since in standard process algebras it would either be modelled using data, e.g. *Buf(q:Queue) := transmit?x:Item; Buf(add(x,q)) + [not(empty(q))] -> receive!first(q); Buf(remove(x))* or using recursion, e.g. *Buf := transmit; receive; stop* ||| *Buf*, neither of which is possible in PEPA. The latter is not possible because all cooperation constructs must appear at the top level in a PEPA specification and *stop* is not an allowed PEPA behaviour.

Timer. The *Timer* monitors the delay between *displays*. Such a feature necessarily requires the *Timer* to "remember"[3] the time of the last *display*, in order to determine whether the next one is on time. The restriction to exponential distributions means that we can only approximate such a feature, which we do using *Erlang* distributions.

An Erlang distribution is a sequence of exponential distributions which approximate a deterministic timing to an arbitrary degree of accuracy [Jai91]. For example, to model an *error* event occurring deterministically at time t, we use a sequence of n *tick* events followed by an *error* event. The *tick* activities are exponentially distributed (rate r_{tick}) where $t = (n \times r_{tick}^{-1}) + r_{error}^{-1}$; this results in a model where the *error* event occurs at time t on average, and the variance of when it occurs gives us the accuracy with respect to timing. We can reduce the overall variance (i.e. increase the accuracy) simply by increasing n and correspondingly increasing r_{tick}.

In our example, we allow *Timer* to *tick* five times before reporting an *error*. It may be reset at any time. Note that it keeps ticking after reporting an *error*, i.e. it is therefore possible to get multiple *errors* before the next frame arrives. We therefore define *Timer* as follows:

$$
\begin{array}{rcll}
Timer & \stackrel{def}{=} & Timer_0 & \\
Timer_0 & \stackrel{def}{=} & (reset, \top).Timer_1 & \\
Timer_i & \stackrel{def}{=} & (tick, r_{tick}).Timer_{i+1} + Timer_0 & 1 \leqslant i \leqslant 5 \\
Timer_6 & \stackrel{def}{=} & (error, r_{error}).Timer_1 + Timer_0 &
\end{array}
$$

4.2 Analysis

Having presented a PEPA description of the basic behaviour of the stream, we can now focus on our main objective: to analyse the quality of service properties of the stream. We will vary parameters in the system and see what consequences they have on the following quality of service properties:-

1. **Latency.** This is *the end-to-end delay between a transmit and its corresponding display*. When deterministic timing is used, the approach is to determine an upper bound on latency, e.g. that the maximum time between generation and display of a frame cannot excede 95ms. Here however, in line with the stochastic approach, we will consider the *average latency*.

2. **Throughput.** We would like to determine the rate at which the *Sink* process displays packets. Clearly, there is a direct link between the rate of loss of the *Medium* and the throughput at the *Sink*. Thus, the flavour of our investigation of this property will be to determine how the rate at which the *Medium* loses messages affects throughput.

3. **Jitter.** Jitter constraints are imposed in order to ensure that there is not an unacceptable variability around the optimum presentation time. In previous work *bounded jitter* has been analysed, i.e. verification has ensured that jitter levels do not stray out of certain upper and lower bounds [BFK+98]. If jitter is bounded in this way then we know that extreme bad (jitter) behaviour cannot occur. However, the resulting constraint is likely to be rather coarse. In particular, extreme fluctuations would be allowed within these bounds. Here we consider a statistical measure of jitter, the *variance of the latency delay*, which yields a more refined jitter property. In the sequel we simply call this *jitter*.

4. **Error Rates.** As discussed earlier our error scenario is that the system simply signals an error, by performing the action type *error*, whenever the gap between consecutive *displays* goes beyond a certain threshold level. We will assess how the rate of these error signals change as we alter other parameters in the system.

[3] We use the term remember in the sense that the timer must *count down* the waiting time in a deterministic fashion. This goes contrary to the memory-less assumption which implies that if an event does not occur in a particular time unit then evaluation of whether it occurs in the next time unit is completely independent of the previous time unit. Thus, the memory-less property implies that there is no sense in which how long a delay has been counting down for is remembered.

To generate meaningful performance figures we analyse the system in its equilibrium state. To do so we build the infinitesimal generator matrix of the corresponding Continuous Time Markov Chain (CTMC). For all states, this matrix gives the probability that the system will be in that state once it has reached equilibrium, i.e. at the steady state. This can be calculated automatically by the PEPA Workbench. To calculate performance figures such as throughput, latency and jitter we need to find the *true rates* of the activities, which in turn requires that we calculate the probability that each activity is enabled.

The system is made up of four processes, and the state of the system changes whenever the state of one of the processes changes. The probability of the system being in a particular state is worked out numerically using MATLAB. The *PEPA State Finder* takes input such as

```
Source_0|*|*|*
```

and returns all the states of the system in which *Source* is in the state $Source_0$. The sum of the probability values of these states is the probability that the *Source* is in state $Source_0$, and we can use this to determine the true rates of components.

4.2.1 True rates and steady state probabilities

Here we show how to derive the various performance measures from the steady state probabilities. We consider $p(Channel_N)$ to be the probability that the *Channel* component of the specification is in state $Channel_N$ at equilibrium, and similarly for *Sink*, *Source* and *Timer*. In addition, $p(Sink_N \textit{ and } Channel_M)$ denotes the probability that the *Sink* component is in state $Sink_N$ and the *Channel* component is simultaneously in state $Channel_M$. These can be determined using the PEPA Workbench, and are used to calculate the true rates of activities.

The specified rate of an activity is not necessarily the same as the rate of that activity in the equilibrium state, since bottlenecks elsewhere in the system may slow the activity down. The true rate (or equilibrium rate) of an activity is thus the specified rate multiplied by the probability that the activity is *enabled*. An activity is enabled if the system is in a state in which it can perform that activity. For example, the true rate of the *display* activity is,

$$true\_r_{disp} = r_{disp} \times \sum_{i=1}^{3} p(Sink_i)$$

since only the *Sink* process is involved in this activity, and it is only capable of performing a display event if it is in one of the states $Sink_1$, $Sink_2$ or $Sink_3$. If r_{loss} is set to zero, then the probability of *Sink* being in state $Sink_1$ ($p(Sink_1)$) is 0.1152, $p(Sink_2) = 0.0136$ and $p(Sink_3) = 0.0016$. So the probability of being in a state where it can perform a *display* is the sum of the above probabilities. Hence the true rate of the display activity is $200 \times (0.1152 + 0.0136 + 0.0016) = 26.0800$ (subject to rounding error, actually 26.0861).

4.2.2 Throughput, latency and jitter

Throughput. The rate of throughput of frames in the equilibrium state is given by the true rate of the *display* activity. This is calculated as shown above.

Latency. Our approach to obtaining the mean end-to-end delay is to sum the mean delays imposed by each individual component in the communication path. To determine the latency of an individual component we must consider the true rates of entry and exit of frames. In our example the precise calculation varies with each component.

The *Source* component does not have an explicit entry activity, since it is modelling the generation of frames. We consider that one frame starts to be formed as soon as the previous one is transmitted, so the latency is given by the mean time between *transmit* activities, which is the inverse of the true *transmit* rate.

$$source\_latency = (true\_r_{trans})^{-1}$$

The *Channel* component poses more problems. We need to take into account the fact that not all frames are passed on to the *Sink*: some are lost via the activity *loss*. The probability of a frame being lost by *Channel* and the probability of it being successfully passed on are determined by the *race condition* between the two activities *loss* and *receive*. If we let *ave_frames_lost* be the average number of frames in the *Channel* which will be lost, and *ave_frames_received* be the average number of frames in the *Channel* which will eventually be received, then

$$ave\_frames\_channel = ave\_frames\_lost + ave\_frames\_received$$

and we have the equality,

$$\frac{ave\_frames\_lost}{ave\_frames\_received} = \frac{true\_r_{loss}}{true\_r_{rec}}$$

Then using Little's law in the context of successful transmissions, the average latency of the successfully passed on frames (*channel_latency*) is given by[4],

$$channel\_latency = \frac{ave\_frames\_received}{true\_r_{rec}}$$

The *Sink* component has only one input and one output activity, and so the latency is given by a straightforward application of Little's Law.

$$sink\_latency = \frac{ave\_no\_frames\_sink}{true\_r_{disp}}$$

The latency of the stream is the sum of the component latencies:

$$stream\_latency = source\_latency + channel\_latency + sink\_latency$$

Jitter. Jitter measures the variability of the time duration between the expected and actual arrival times of packets. This will be the variance of a sum of exponential distributions, one for each component in the system. Since these distributions are all independent, the variance of the sum is simply the sum of the variances (see [HP93]), i.e.

$$jitter = source\_variance + channel\_variance + sink\_variance$$

where, for example,

$$source\_variance = (true\_r_{trans})^{-2}$$

4.2.3 Component usage

We can also determine the average number of frames in a component by taking a weighted sum of the appropriate probabilities. For example, the average number of frames in the *Channel* component is

$$\sum_{i=0}^{5} i \times p(Channel_i)$$

In a similar fashion we can calculate the average number of frames in the *Source* and *Sink* components, and the average number of frames in the entire system is the sum of these averages.

We can also calculate idling and busy times: the percentage of time that a component spends idling is given by the probability that there are no frames in the component. The percentage busy time is the probability that there are one or more frames in the component.

r_{loss}	00.0	10.0	20.0	30.0	40.0	50.0
$true\_r_{loss}$	00.0	9.5490	18.0536	25.0964	30.6408	34.9059
$true\_r_{trans}$	29.0897	37.7430	44.7389	49.8593	53.3447	55.6212
$true\_r_{rec}$	29.0897	28.1940	26.6853	24.7629	22.7039	20.7153
$true\_r_{disp}$	29.0897	28.1940	26.6853	24.7629	22.7039	20.7153
$true\_r_{tick}$	99.4546	99.4455	99.4287	99.4056	99.3795	99.3531
$true\_r_{error}$	10.9080	11.0891	11.4261	11.8875	12.4098	12.9375
ave no. in source	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ave no. in chan	4.1273	3.6087	3.0477	2.5203	2.0715	1.7112
ave no. in sink	0.1713	0.1648	0.1545	0.1418	0.1285	0.1160
ave no. in stream	5.2985	4.7735	4.2021	3.6620	3.2000	2.8272
source latency (s)	0.0344	0.0265	0.0224	0.0201	0.0187	0.0180
chan latency (s)	0.1419	0.0956	0.0681	0.0505	0.0388	0.0308
sink latency (s)	0.0059	0.0058	0.0058	0.0057	0.0057	0.0056
stream latency (s)	0.1821	0.1280	0.0963	0.0763	0.0632	0.0543
throughput jitter	0.0012	0.0013	0.0014	0.0016	0.0019	0.0023
channel jitter	0.0012	0.0013	0.0014	0.0016	0.0019	0.0023
source jitter	0.0012	0.0007	0.0005	0.0004	0.0004	0.0003
latency jitter	0.0035	0.0032	0.0033	0.0037	0.0042	0.0050

Figure 2: Table 1 - Analysis of Stream

4.3 An Example

With the PEPA Workbench, we can calculate the various performance figures and quality of service parameters we are interested in. For example, with the following particular rates: $r_{trans} = 60.0$; $r_{rec} = 30.0$; $r_{disp} = 200.0$; $r_{tick} = 100.0$; $r_{error} = 2000.0$; $r_{reset} = 2000.0$ and varying r_{loss} we get the table shown in Figure 2.

In explaining this table we can make a number of points:

1. As the rate of *loss* increases the true rate of transmission increases (since the *Channel* is less often full); the true rate of transmission tends to the specified rate of transmission, i.e. 60, as r_{loss} tends to infinity.

2. The true rates of reception and *display* are equal, since no frames are lost between these activities and the true rates of reception and *display* decrease as *loss* increases, for obvious reasons.

3. The true rate of the *tick* event does not change greatly when the rate of *loss* is increased. This is because of the use of the Erlang distribution, i.e. the large number of *tick* events ensures that *tick*'s are "almost" independent of *reset* events. In addition, *reset* and *error* events are very fast events relative to *tick*.

4. The results here allow us for example to relate the rate of *loss* to the throughput. For example, if we wished to ensure that the throughput (true rate of *display*) was greater then 28 packets per second then we would know that setting the rate of *loss* to 10.00 would be close to the boundary condition.

5. The true rate of *display* is very different to the specified rate of *display*. This is because the *Sink* needs something to *display* before it can do anything, i.e. it spends much of its time in state $Sink_0$.

[4] In fact, because of the assumptions implicit in Markovian analysis, this turns out to be equal to the latency of the lost frames.

6. The average number of frames in the stream declines as the rate of *loss* increases, for obvious reasons. In addition, latency of the stream component and the stream itself decrease as the rate of *loss* increases.

4.4 Figures for the Tempo Stream

The example that we have analysed here is based upon previous formulations of the problem to be found in [BBBC93, Bla94, BBBC98]. In this section we investigate to what extent we can bring our analysis in to line with the specification to be found in [Bla94]. One reason for doing this is to make the results of our analysis relevant to the earlier work, thus enabling our results to inform those found in [Bla94]. We inform the earlier work in two ways, firstly by providing a formal analysis ([Bla94] just gives a specification of the problem) and secondly, because our analysis is performed in a stochastic context, [Bla94] only considers deterministic timings.

In pursuing this goal, we firstly, in line with the specification in [Bla94], employ a marginally more sophisticated *Source* process:-

$$\begin{aligned} Source_0 &\stackrel{def}{=} (gen, r_{gen}).Source_1 \\ Source_1 &\stackrel{def}{=} (transmit, r_{trans}).Source_0 \end{aligned}$$

which differentiates between the generation of frames (the *gen* activity) and the transmission of frames (the *transmit* activity). Secondly, we have attempted to bring the figures resulting from our analysis into line with those used in [Bla94]. The requirements given in [Bla94] are:

- The data source generates frames at a rate of 30 frames per second.
- After generation, 5ms elapse before it is transmitted
- Successfully transmitted frames arrive at the data sink between 15ms and 20ms after transmission
- The data sink takes 5ms to process a frame
- The end-to-end latency of a single frame should not exceed 30ms
- The end-to-end throughput should be within 25 and 35 frames per second.

In attempting to follow these figures we obtained the table shown in figure 3, where $r_{gen} = 35.3$; $r_{trans} = 200.0$, $r_{rec} = 78.0$, $r_{disp} = 200.0$, $r_{tick} = 50.0$, $r_{error} = 2000.0$ and $r_{reset} = 2000.0$. We can see from the table that using these parameters enables us to model the requirements given in [Bla94], which were highlighted above. In particular, the figures found in the first two columns in this table fall within the required timings. This is subject to the fact that we are working with average latency values rather than crude latency bounds. Thus, the first column, where loss is zero, probably has too high an end-to-end latency value: 29.99 ms, i.e. since variance of latency (jitter) is non-zero some transmissions will certainly invalidate the 30ms upper bound on end-to-end latency.

Thus, the second column contains figures that are probably most closely in line with those in [Bla94]. Focussing on this column, we can identify a number of conclusions, which inform the earlier multimedia stream work. Firstly, the figures identify an acceptable bound on *loss* (i.e. a true rate of 3.4488) and indicate a certain rate of *error* (i.e. a true rate of 3.2493).

Furthermore, the analysis reveals that the average number of frames in the stream at any one time is never more than one and as the rate of loss increases this number declines. This indicates that the requirements given in [Bla94] are not completely realistic; in particular that the channel itself is not accurately modelled. Two possible ways of improving the modelling are allowing multiple sources and sinks to use the same channel; and modelling the channel as a sequence of buffers, each of which delays the frames as they pass through.

r_{loss}	00.0	10.0	20.0
$true\_r_{loss}$	00.0	3.4488	6.1772
$true\_r_{gen}$	30.0041	30.0046	30.0042
$true\_r_{trans}$	30.0041	30.0046	30.0042
$true\_r_{rec}$	30.0041	26.5554	23.8270
$true\_r_{disp}$	30.0041	26.5554	23.8270
$true\_r_{tick}$	49.9294	49.9188	49.9090
$true\_r_{reset}$	30.0041	26.5554	23.8270
$true\_r_{error}$	2.8233	3.2493	3.6395
ave no. in source	0.1500	0.1500	0.1500
ave no. in channel	0.5742	0.4735	0.4035
ave no. in sink	0.1736	0.1505	0.1329
ave no. in stream	0.8978	0.7740	0.6865
source latency (s)	0.0050	0.0050	0.0050
channel latency (s)	0.0191	0.0158	0.0134
sink latency (s)	0.0058	0.0057	0.0056
stream latency (s)	0.0299	0.0264	0.0240
variance of sink	0.0011	0.0014	0.0018
variance of channel	0.0011	0.0014	0.0018
variance of source	0.0011	0.0011	0.0011
jitter	0.0033	0.0039	0.0046

Figure 3: Table 2 - Tempo like figures

5 Immediate Actions

In this section we consider to what extent immediate actions influence the analyse we obtained above. As suggested earlier, it may be possible to reduce the variance of the error action by using immediate actions, and some work has been done on including immediate actions in stochastic process algebra. In [HRW95], immediate actions are added to a basic stochastic process algebra. The resulting language is called TIPP and it extends the class of processes which may be specified. But in order to derive a Continuous Time Markov Chain immediate actions must have only an internal impact, and to capture this an equivalence, called Markovian Observational Congruence, is defined. Every term in the TIPPtool input language [KM98] can be interpreted as a Continuous Time Markov Chain, provided all delays are Markovian. The TIPPtool allows CTMC analysis similar to the capabilites of the PEPA workbench.

Timeouts are approximated by Erlang distributions followed by immediate actions. Thus if, in a similar way to in TIPP, we could use immediate actions, then we could model *error* as a visible immediate action, and we could define *Timer* as

$$\begin{aligned}
Timer_0 &\stackrel{def}{=} (reset, \top).Timer_1 \\
Timer_i &\stackrel{def}{=} (tick, r_{tick}).Timer_{i+1} + Timer_0 \qquad 1 \leqslant i \leqslant 5 \\
Timer_6 &\stackrel{def}{=} error.Timer_1 + Timer_0
\end{aligned}$$

However, the equational laws of Markovian Observational Congruence, to be found in [HRW95], give us

$$Timer_6 \quad = \quad error.Timer_1 + Timer_0$$

$$= \quad error.Timer_1 \qquad\qquad (Axiom9)$$

which reflects the fundamental property of immediate actions: that they always "win" the race condition. Furthermore, we get that

$$Timer_5 \quad = \quad (tick, r_{tick}).error.Timer_1 + (reset, \top).Timer_1$$

and so the only difference here is that when the *error* is enabled, it has to happen immediately.

So, in a stochastic process algebra which provides them, we can use immediate actions to signal errors. However, in a situation where an Erlang distribution has been used to approximate a deterministic delay, making the *error* an immediate action will only have a very minor impact on the error variance. To see this, consider the example of the *Timer* above. Error variance is calculated as

$$errorvariance = 5 * (1/(true\_r_{tick} * true\_r_{tick})) + (1/(true\_r_{error} * true\_r_{error}))$$

With the rate of the error action set to 2000, the error variance is 0.0020, and with the rate of the error action set to 200000, the error variance is also 0.0020.

It is evident from these figures that once the rate of error is sufficiently fast, increasing it does not alter the variance significantly. The Erlang distribution itself is responsible for all the variance.

In conclusion, although in an appropriate SPA we could specify the multimedia stream using an immediate action for the error, since it would make no difference to the performance figures presented in this paper we have not followed this route.

6 Assessment and Further Work

6.1 Assessment of PEPA

This subsection gives a short assessment of PEPA (and stochastic process algebra in general) in the light of our application of them to specifying and analysing the multimedia stream. Our experience with PEPA has generally been positive. Its major strength being that it supports automated analysis and corresponding generation of performance figures. This is a major strength of the technique. Clearly, restricting to exponential distributions is critical in enabling such analysis to be performed.

A number of limitations of the approach can also be highlighted. These typically reflect the current "state of the art" of stochastic process algebra techniques.

- *Change of Mind Set.* Specification in PEPA requires a significant change of mind set from specification in classic process algebra, such as CCS [Mil89], CSP [Hoa85] and LOTOS [BB88]. A central aspect of this change is the nature of action offers. The classic process algebra interpretation is that actions are offered to the environment, which decides whether to take them. Thus, in this aspect, the system is *passive*[5] - the system offers a set of actions, then it waits passively for the environment to decide which (if any) to take. (Deterministically) timed process algebras, such as Timed CSP [Dav93] or ET-LOTOS [LL93], refine this interpretation by allowing time bounds to be placed on the period of time in which actions are (passively) offered to the environment; untimed process algebra can be seen as a subclass of timed process algebra where the time bounds are always zero to infinity.

  In PEPA the interpretation is somewhat different. Firstly, the basic unit of modelling is an activity, the completion of which is marked by the occurrence of an action type. Importantly, although the occurrence of this action type can be seen by the environment, it is not directly controlled by the

[5]Internal actions complicate this interpretation, since their selection is determined internally by the system. Thus, what we say largely concerns observable actions.

environment. In this way, the system is more *active* in deciding the instance of action occurrence, this is born out by the discussion in chapter 3 of [Hil96]. In fact, the PEPA interpretation is one of usage of (implicit) resources. Thus, choice models competition for a resource while parallel composition represents cooperative use of resources in performing activities.

This change of mind set can be difficult to come to terms with when starting to use PEPA. Also, for some specification problems both the classic interpretation and the PEPA interpretations can arise in describing the same system.

- *Deadlock States.* Another aspect of moving from the classic process algebra model to PEPA is that, in order to enable Markovian analysis to be performed, deadlocks cannot arise in the system specification. A consequence of which is that the the deadlock process *stop* does not appear in the PEPA abstract syntax. In our case study this became a problem when we tried to describe error behaviour, i.e. we would have liked to have allowed the system to time out and then stop. With respect to this problem, a possible area for future work is *transient analysis*, which determines the probabilities of being in particular states before equilibrium is reached. There are a number of numerical methods which can be used to find transient solutions to Markov chains (see for example [Ste94]). In addition, the TIPPtool [KM98] allows transient analysis - if the labelled transition system generated from a specification is not strongly connected, a time instant can be given to the tool and it will compute the probabilities of being in particular states at that time.

- *Setting True Rates.* A useful feature would be the ability to set the true rate of a particular transition, i.e. the analysis would ensure that the rate specified for a particular transition is indeed its true rate and would adjust the true rates of other activities accordingly. This would, for example, have enabled us to set the true rate at which frames are *transmit*ted and see how other parameters vary around this rate. Thus, such a feature would have been useful when trying to relate the results of our analysis to the earlier stream specifications.

- *Deterministic Timing.* It is clear from our case study that even in the context of stochastic specification, deterministic timings will frequently arise. Modelling a timeout from which an error state is reached is an example which arises in our specification. In a Markovian setting, the standard solution is to use an Erlang distribution, as we have indeed done. This is a reasonable solution, however, it potentially leads to a massive state explosion, which would prohibit the application of support tools. The state explosion is contained in our application since we only have a single Erlang distribution. However, if a number of Erlang distributions evolve concurrently, their component phases are interleaved, which causes state explosion according to the product of the number of phases.

- *Generalised Distributions.* The last point leads onto what is perhaps the most fundamental limitation of the PEPA approach, and that is what is also its strength - the restriction to exponential distributions. Generalised distributions are required, not just in order to obtain deterministic timing, but since distributions found in the application area commonly fail to be memoryless (or deterministic). For example, in our case study, the rate of the action *receive* has a major affect on determining the latency delay of the channel and this rate is assumed to be exponentially distributed. However, it is well known that packet lengths are not in reality exponentially distributed, rather they are either of constant length (as in ATM cells [Tan96]) or they are uniformly distributed with minimum and maximum size (as in Ethernet frames [Tan96]). Furthermore, the latency delay imposed by a channel will clearly be tied to packet lengths. Thus, our assumption of an exponential channel latency is not in practice realistic.

  This observation suggests that a suitable modelling technique should support generalised distributions. This brings a number of problems, not least of which is that analytical techniques become significantly more complicated [Kin90]. In addition, it has been pointed out [Kat96] that use of exponential distributions is very closely tied to the interleaving assumption underlying parallel composition in process algebra. Furthermore, it is suggested [Kat96] that true concurrency models, which are typically more complex than interleaved approaches, are appropriate to be used in the presence of generalised distributions.

6.2 Further work

The assessment made in the previous subsection suggests a number of areas for future work. Firstly, we are investigating the applicability of transient analysis to our case study. This is being performed in the context of an assessment of the TIPP approach [HRW95]. In addition, we are exploring a number of approaches that support generalised distributions, e.g. SPADES [DKB97]. We are also working on model checking techniques in a stochastic setting [ACD91] and we intend to analyse some larger multimedia case studies, e.g. the lip synchronisation specification to be found in [BBBC98].

Acknowledgements

We would like to thank Joost-Pieter Katoen and Holger Hermanns from the University of Erlangen, who fielded a number of our questions on stochastic process algebras. Also, Stephen Gilmore from the University of Edinburgh advised on using PEPA, while Lynne and Gordon Blair from Lancaster University were involved in V-QoS discussions from which this paper has grown. Joost-Pieter Katoen and Lynne Blair provided valuable comments on a draft of the paper.

References

[ABSS96] A. Feyzi Ates, M. Bilgic, S. Saito, and B. Sarikaya. Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE Journal on Selected Area in Communications*, 14:126–137, 1996.

[ACD91] Rajeev Alur, Costas Courcoubetis, and David Dill. Verifying automata specifications of probabilistic real-time systems. In *Proceedings of Real-Time Theory in Practice*, Lecture Notes in Computer Science 600, pages 28–44, 1991.

[BB88] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

[BBBC93] H. Bowman, L. Blair, G. S. Blair, and A. Chetwynd. Time versus abstraction in formal description. In *FORTE'93*, Boston, October 1993. North-Holland.

[BBBC98] G.S. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specification of Distributed Multimedia Systems*. University College London Press, 1998.

[BDG95] Marco Bernardo, Lorenzo Donatiello, and Roberto Gorrieri. Integrating performance and functional analysis of concurrent systems with empa. Technical Report UBLCS-95-14, Department of Computer Science, University of Bologna, Piazza di Porto S. Donato, 5, 40127 Bologna, September 1995. Revised March 1996.

[BFK+98] H. Bowman, G. Faconti, J-P. Katoen, D. Latella, and M. Massink. Automatic verification of a lip synchronisation algorithm using UPPAAL. 1998. Submitted for publication.

[BFM98] H. Bowman, G. Faconti, and M. Massink. Specification and verification of media constraints using UPPAAL. In *5th Eurographics Workshop on the Design Specification and Verification of Interactive Systems, DSV-IS 98, Abingdon, UK*, Eurographics Book Series. Springer-Verlag, 1998.

[Bla94] Lynne Blair. *The Formal Specification and Verification of Dsitributed Multimedia Systems*. PhD thesis, Lancaster University, September 1994.

[Dav93] Jim Davies. *Specification and Proof in Real-time CSP*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1993.

[DKB97] Pedro R. D'Argenio, Joost-Pieter Katoen, and Ed Brinksma. A stochastic automata model and its algebraic approach. In Ed. Brinksma and Albert Nymeyer, editors, *Process Algebra and Performance Modelling. Fifth International Workshop*, number 97-14 in CTIT Technical Report, pages 1–16, P.O. BOX 217 - 7500 AE Enschede, The Netherlands, June 1997. University of Twente.

[FL98] S. Fischer and S. Leue. Formal methods for broadband and multimedia systems. *Computer Networks and ISDN Systems, Special Issue on Trends in Formal Description Techniques and their Applications, to appear*, 1998.

[Gil97] Stephen Gilmore. *The PEPA Workbench: Users Manual.* University of Edinburgh, April 1997. `http://www.dcs.ed.ac.uk/pepa`.

[HCCB94] D. Hutchison, G. Coulson, A. Campbell, and G.S. Blair. *Quality of Service Management in Distributed Systems.* Addison Wesley, 1994.

[Hil96] Jane Hillston. *A Compositional Approach to Performance Modelling.* Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[HP93] Peter G. Harrison and Naresh M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* International Computer Science Series. Addison-Wesley, 1993.

[HRW95] Holger Hermanns, Michael Rettlebach, and Thorsten Weiss. Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching. *The Computer Journal*, 38(7):530–541, 1995.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis.* J. Wiley, New York, 1991.

[Kat96] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, Centre for Telematics and Information Technology, P.O. Box 217, 7500 AE Enschede The Netherlands, April 1996.

[Kin90] Peter J. B. King. *Computer and Communications Systems Performance Modelling.* Prentice Hall International Series in Computer Science. Prentice Hall, UK, 1990.

[KM98] U. Klehmet and V. Mertsiotakis. TIPPtool: Timed Processes and Performability Evaluation. Technical Report 1/98, University of Erlangen, 1998.

[LL93] Luc Leonard and Guy Leduc. An enhanced version of timed LOTOS and its application to a case study. In *FORTE'93*, Boston, October 1993. North-Holland.

[MBC+95] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley, 1995.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Reg93] T. Regan. Multimedia in temporal LOTOS: A lip synchronisation algorithm. In *PSTV XIII, 13th Protocol Specification, Testing and Verification.* North-Holland, 1993.

[Slo94] M. Sloman, editor. *Network and Distributed Systems Management.* Addison Wesley, 1994.

[Ste94] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princetown University Press, Princetown, New Jersey, 1994.

[Tan96] A. S. Tanenbaum. *Computer Networks (3rd Edition).* Prentice-Hall, 1996.