Guaranteeing Real-Time Traffic Through an ATM Network[1]

Hans Hansson Mikael Sjödin Ken Tindell
Dept. of Computer Systems, Uppsala University
P.O. Box 325, S-751 05 Uppsala, Sweden
e-mail: {hansh,mic,ken}@docs.uu.se

Abstract

We present a method for providing hard real-time guarantees for traffic through an Asynchronous Transfer Mode (ATM) network. Guaranteed delivery also implies that we have to guarantee that no cells are lost due to over-full buffers in the network.

By using priority queues in the output buffers we allow urgent messages short end-to-end delays, while still guaranteeing end-to-end delays for low-priority messages.

We can determine á priori *if message deadlines will be met by calculating maximum end-to-end delays and buffer-needs. Analysis is based on CPU schedulability analysis, which means that it provides a framework for analysing an entire system, including both CPU and network scheduling.*

1. Introduction

Asynchronous Transfer Mode (ATM) is an emerging technology for telecom and data networking [8]. In this paper we present a method for providing *hard real-time* guarantees for traffic through an ATM network. We provide analysis that can be used by an *admission control* procedure to determine if a specified maximum transmission delay (latency) for a new data stream can be guaranteed or not. In addition, the analysis detects if requirements on already admitted streams are violated, or if buffer overflows can occur when the new stream is admitted.

It is important that solutions to the real-time admission control problem can be easily incorporated with existing ATM standards. It must also be possible to make efficient implementations, supporting the very high bandwidth of current and future ATM networks. Furthermore, an admission control scheme must allow for high utilisation of network resources. Since not all streams may require absolute guarantees it should be possible to integrate hard real-time traffic with other ATM *Quality of Service* (QoS) classes which do not require absolute guarantees on the timely delivery of messages.

[1] This work has been supported by the Swedish board for technical development (NUTEK), within the Swedish ATM Platform consortium (SWAP).

An additional benefit would be if the analysis could be incorporated with an existing framework for real-time systems analysis. This would allow system designers to use the same technique for assessing the timely behaviour of the entire system, including both CPU and network scheduling.

In this paper we propose the use of priority queueing, together with response-time analysis for calculating the worst case end-to-end delays in the network. In combination with buffer-need analysis we have a sufficient method for providing deterministic guarantees on the delivery of real-time messages. The required priority management and queueing are simple to implement; indeed, today many modern ATM switches support priority queueing, e.g. Fore Systems ASX-200BX, ASX-1000 and Cisco LS 1010.

The motivation for our work is that statistical guarantees, e.g. in terms of "message loss probability" or "average message delay", are not sufficient for many applications, e.g. vehicle and industrial control systems. Instead, a firmer commitment is needed from the network, saying that it will *always* deliver *all* messages *completely* and *within* specified deadlines (with reservation for hardware failure and external physical interference, both which are very unlikely in fibre optical networks).

It is likely that ATM will be used in *safety critical* applications, simply because it will be the cheapest high bandwidth communication technology available. Also, ATM is an attractive solution in industrial environments due to its optical signalling, avoiding electrical signalling which in these harsh environments may be error prone.

A further motivation for our work is that an admission control scheme providing absolute guarantees can be used for traditional *soft real-time* communication, such as multimedia, provided that the scheme gives performance comparable to (or better than) statistical methods.

Outline: In section 2 we give a brief introduction to ATM networking, in section 3 we present our traffic and network models. Sections 4 and 5 present the response-time and buffer-need analysis. In section 6 we present analysis of some sample networks and traffic situations.

Finally, section 7 relates our work to other research and provides some concluding remarks.

2. Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) [8] is a high performance, connection oriented, network architecture. Over each connection a *stream* of fixed size packets, called *cells*, are transmitted. Each cell is 53 bytes long, 5 byte header information + 48 byte *payload* (user data). Long messages are segmented and sent in consecutive cells in the stream. The ATM network guarantees that cells within a stream is delivered in the same order as they were transmitted.

An ATM network consists of a set of *ATM switches*. Each switch routes cells from a set of *input ports* to a set of *output ports*, based on stream identifiers in the cell-headers. At the output port of an ATM switch a *port controller* multiplexes cells from different streams onto the outgoing lines.

ATM has been designed to support the extremely high bandwidth and optical signalling, of todays' and future's information super highways. Hence, it will handle a number of communications services, including phone calls, video conferencing and computer communication. ATM is also proposed as a Local Area Network (LAN).

Clearly, fulfilling all these requirements is not an easy task. Different applications have different timing and bandwidth requirements. To make sure that each application gets required Quality of Service (QoS), the network must perform some form of admission control on new streams. The admission control should ensure that there are enough resources in the network to give the new stream its required QoS, without compromising the QoS of already admitted streams.

3. Our Model

We present a scheme where *Fixed Priority Scheduling* (FPS) is used to schedule the departure of cells. This means that a fixed priority is associated to each stream and cell.

We assume that the switch and the source nodes send cells in priority order and that cells with the same priority are served in FIFO order.

We further assume that the source nodes have some form of rate control which releases cells within one message with a fixed rate, e.g. using a *leaky bucket* [15]. Such rate-control is a standard technique to decrease the buffer-need, and interference between different traffic streams.

3.1. The Network

In this paper we consider the single switch network depicted in figure 1. The analysis can however be extended to cover multi-switch networks. In figure 1, each in-bound line to the switch is connected to a *source* node and each out-bound line is connected to a *destination* node. Each source (destination) can be the start (end) point of several streams. With each stream we associate a traffic generating process ("TG" in figure 1).

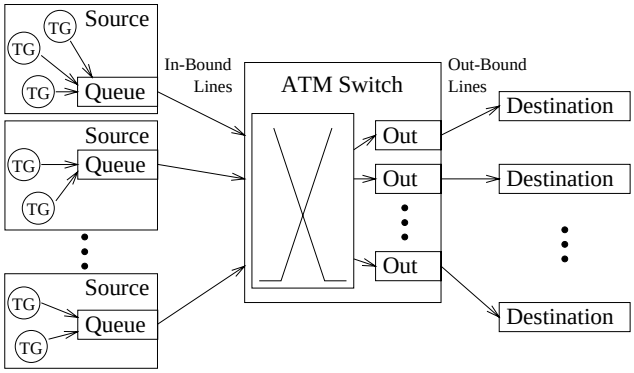


Figure 1: A one switch network

Each in- or out-bound line l has the following timing attributes:

$tTrans_l$ the transmission time of one cell on the line, i.e. the time to put a cell onto the line (this relates to the bandwidth of the line),

$tProp_l$ the time for a signal to propagate between the end-points of the line (this relates to the length and type of physical media used), and

$SendT_l = tTrans_l + tProp_l$ the time between the start of transmission of a cell and the time when the cell has fully arrived at the other end of line l.

The switch is assumed to:

- be output buffered with one priority queue for each output port ("Out" in figure 1),
- use a non-blocking switch architecture (i.e. cell loss or cell contention may only occur at the output buffers), and
- be able to store and retrieve the priority of each stream.

The timing properties of the switch are expressed by the two attributes: $tSwitch^{max}$ and $tSwitch^{min}$, representing the maximum and minimum time from when a cell has fully arrived at an input port to when it has been placed in its output queue.

Using these attributes different switch architectures can be modelled. In [17] we describe how to model a Time Division Multiple Access (TDMA) switch (in which each input port is serviced in round Robin order and one cell at a time is switched from input port to output port).

3.2. The Traffic

The traffic consists of a set of streams, S. Each stream, $i \in S$, is characterised by the following attributes:

T_i The *period* (a strictly positive rational number) of message queueings. We do not require strict periodicity, instead the period should be interpreted as a required minimum message inter-queueing time.

t_i The time between queueing of consecutive cells, i.e. the leaky bucket rate ($t_i > 0$).

$J_{(i,0)}$ The initial *jitter* (i.e. the variance in time) on the queueing of messages, $J_{(i,0)} \geq 0$. The initial jitter tells how early (with respect to T_i) a message can arrive from the application to the source output queue. (If response-time analysis is used to analyse the processes generating the traffic, $J_{(i,0)}$ can be derived from that analysis [19]).

D_i The *deadline*, expressing the maximum allowed delay for delivery of a message in stream i.

N_i the maximum number of cells needed to transfer a message. If the maximum message size of stream i is s_i bytes then $N_i = \lceil \frac{s_i}{48} \rceil$. (Where $\lceil\ \rceil$ denotes the *ceiling* operator returning the smallest integer equal to or grater than its operand.)

P_i The *priority*. If stream i has higher priority than j then $P_i > P_j$. Every stream is assumed to have a unique (and fixed) priority.

L_i A vector identifying the lines used along the path of stream i. $L_{(i,1)}$ identifies the line between the source and the switch, while $L_{(i,2)}$ identifies the line between the switch and the destination.

4. Response-Time Analysis

We will now for the above model formulate response-time analysis equations which for each stream can be used to calculate the maximum end-to-end message delay.

We divide the total maximum delay into five delays on the path of each stream. The delays are (see figure 2):

$Q_{(i,1)}$, the delay in the priority output queue in the source,

$SendT_{L_{(i,1)}}$, the delay in the line between the source and the switch,

$tSwitch^{max}$, the delay in the switching fabric,

$Q_{(i,2)}$, the delay in the priority queue at the output port in the switch, and

$SendT_{L_{(i,2)}}$, the delay in the line between the switch and the destination.

$Q_{(i,h)}$ denotes the maximum queueing-delay for a cell in stream i at hop number h.

We let Msg_i^{max} and Msg_i^{min} denote the maximum and minimum response-times for a message in stream i.

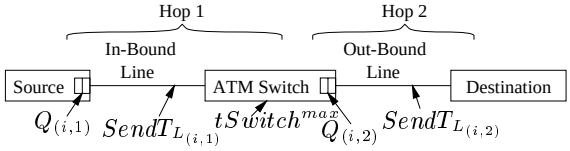


Figure 2: The five delays in a one switch ATM network

The equations to calculate them are

$$
\begin{aligned}
Msg_i^{max} &= \underbrace{t_i * (N_i - 1)}_{\text{Time to pass through the leaky bucket}} + R_i^{max} \\
Msg_i^{min} &= R_i^{min} \\
J_i^{msg} &= Msg_i^{max} - Msg_i^{min} + J_{(i,0)}
\end{aligned}
$$

J_i^{msg} is the message jitter in the reception of messages at the destination node. R_i^{max} and R_i^{min} denotes the maximum and minimum response-times for one cell in stream i.

$$
\begin{aligned}
R_i^{max} &= R_{(i,1)}^{max} + R_{(i,2)}^{max} \\
R_{(i,1)}^{max} &= Q_{(i,1)} + SendT_{L_{(i,1)}} + tSwitch^{max} \\
R_{(i,2)}^{max} &= Q_{(i,2)} + SendT_{L_{(i,2)}} \\
R_i^{min} &= R_{(i,1)}^{min} + R_{(i,2)}^{min} \\
R_{(i,1)}^{min} &= SendT_{L_{(i,1)}} + tSwitch^{min} \\
R_{(i,2)}^{min} &= SendT_{L_{(i,2)}} \\
J_{(i,h)} &= R_{(i,h)}^{max} - R_{(i,h)}^{min} + J_{(i,h-1)}
\end{aligned}
$$

Where $R_{(i,h)}^{max}$, and $R_{(i,h)}^{min}$, denote the maximum and minimum response-time for a cell in stream i at hop h. $J_{(i,h)}$ denotes the *accumulated jitter* for a cell in stream i after hop h. To complete the definition above we must define the maximum queueing-delay $Q_{(i,h)}$.

4.1. Queueing-Delay

The delay $Q_{(i,h)}$ is the maximum queueing-delay for a cell in stream i at hop h.

Using response-time analysis we can calculate the maximum queueing-delay for the k-th cell of $k+1$ queued cells in stream i under a *busy period* where no more than one lower priority cell is dequeued. We use $Q_{(i,h)}^{:0\cdots:k}$ to denote the dequeuing-time of the k-th cell (the first cell is cell number $k = 0$).

We use $A_{(i,h)}^{:k}$ to denote the earliest arrival (i.e. queueing-time) of cell k in stream i. If the $k+1$-th cell in stream i arrives before the dequeuing-time of the k-th cell, i.e. if $Q_{(i,h)}^{:0\cdots:k} > A_{(i,h)}^{:k+1}$ then the busy period will last for at least $k+1$ cells. To find the maximum queueing-delay we need to consider queueing-delays for each cell in stream i queued during the busy period.

We can define the maximum queueing-delay for cell k as $Q_{(i,h)}^{:k} = Q_{(i,h)}^{:0\cdots:k} - A_{(i,h)}^{:k}$, i.e. the difference between

the latest dequeuing-time and the earliest queueing-time.

The maximum queueing-delay for any cell in stream i at hop h is then given by $Q_{(i,h)} = \max_k Q^{:k}_{(i,h)}$.

Figure 3 shows the earliest queueing-times for cells from the two first messages in a stream i where $N_i = 4$. In the figure $j_{(i,h)}$ denotes the jitter incurred by the network after h hops (i.e. the jitter added to the initial jitter), defined as $j_{(i,h)} = J_{(i,h)} - J_{(i,0)}$.

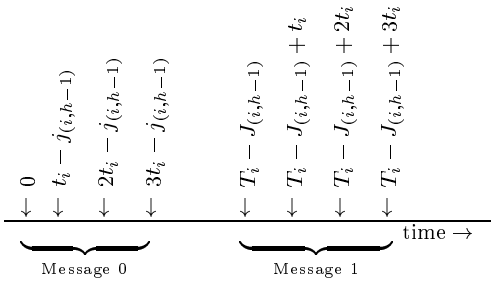


Figure 3: The earliest arrival times of cells

In the worst case scenario, cells from the second message will start to arrive to the queue at time $T_i - J_{(i,h-1)}$. Note that $J_{(i,h-1)}$ includes $j_{(i,h-1)}$ thus $j_{(i,h-1)}$ is not explicitly considered when determining the arrival times of cells belonging to messages other than the first.

The behaviour in figure 3 is captured by the following definition of arrival times

$$A'^{:k}_{(i,h)} = \begin{cases} 0 & \text{if } k = 0 \\ k * t_i - j_{(i,h-1)} & \text{if } 0 < k < N_i \\ M_k * T_i - J_{(i,h-1)} + m_k * t_i & \text{if } k \geq N_i \end{cases}$$

$$M_k = \left\lfloor \frac{k}{N_i} \right\rfloor$$

$$m_k = k - N_i * M_k$$

Where M_k denotes the number of cells in k full messages, and m_k is the number of cells sent from the last message. $\lfloor\ \rfloor$ is the *floor* operator, returning the largest integer equal to or less than its operand.

However, since there is no limit on the jitter we might have a situation where $j_{(i,h)} > t_i$, i.e. in the model there is a risk that cell $k+1$ arrives before cell k. Since this cannot occur in the real system we need to capture the fact that cells arrive in the same order as they are sent. Cell $k+1$ cannot arrive sooner than $tTrans_{L_{(i,h-1)}}$ after cell k. Thus $A^{:k}_{(i,h)}$ becomes

$$\begin{aligned} A^{:0}_{(i,h)} &= 0 \\ A^{:k}_{(i,h)} &= max(A^{:k-1}_{(i,h)} + tTrans_{L_{(i,h-1)}}, A'^{:k}_{(i,h)}) \end{aligned}$$

Note that $L_{(i,0)}$, is undefined. For sources we have no line which delivers cells, thus there is no minimum inter-arrival time for cells at the source. We can model this by defining $L_{(i,0)}$ to be a (fictitious) line with $tTrans_{L_{(i,0)}} = 0$

We define $Q^{:0\cdots:k}_{(i,h)}$ as follows

$$Q^{:0\ldots k}_{(i,h)} = (k+1) * tTrans_{L_{(i,h)}} + \sum_{j \in hp_{L_{(i,h)}}(i)} I_j * tTrans_{L_{(i,h)}}$$

$$I_j = \underbrace{W_{(j,h)}(Q^{:0\ldots k}_{(i,h)}) * N_j}_{\text{Whole messages}} + \underbrace{min\left(\left\lceil \frac{Q^{:0\ldots k}_{(i,h)} + J_{(j,h-1)} - W_{(j,h)}(Q^{:0\ldots k}_{(i,h)}) * T_j}{t_j} \right\rceil, N_j\right)}_{\text{Partial message}}$$

$$W_{(j,h)}(t) = \left\lfloor \frac{t + J_{(j,h-1)}}{T_j} \right\rfloor$$

- The term $(k+1) * tTrans_{L_{(i,h)}}$ represents the time to send one low-priority cell and the k first cells in stream i.
- $hp_l(i)$ denotes the set of streams using line l ($L_{(i,h)}$) and having higher priority than P_i.
- The interference term, I_j, denotes the number of cells a higher priority stream j is putting into the queue during the queueing-delay of the cells in stream i at hop h.
- In the definition of I_j we have a certain number of cells from fully queued messages and possibly some cells from a partially queued message.
- $W_{(j,h)}(t)$ denotes the number of whole messages queued in stream j at hop h during and interval of length t.
- $J_{(j,h-1)}$ in I_j and $W_{(j,h)}$ represents that due to jitter the two first messages of a stream in the worst case scenario are separated by $T_j - J_{(j,h-1)}$ instead of a full period T_j. Thus, causing a little more interference than if all messages were separated by T_j.

Since $Q^{:0\ldots k}_{(i,h)}$ occurs on both sides of the equality but inside $\lfloor\ \rfloor$ and $\lceil\ \rceil$ operators the equation cannot be solved algebraically. However, it is straightforward to solve this type of fixpoint equations numerically [4].

5. Buffer-Need Analysis

To guarantee that no cells are lost due to buffer overflow we need to analyse the buffer-need and ensure that it never exceeds the available buffer space. In this section we will derive the maximum buffer-need for each output port.

We will assume that response-time analysis has been performed and that all streams meet their deadlines. This implies that each line is utilised less than 100%.

Consider one output port. We use o to denote the line connected to this port. S is the set of all streams, and $S_o \subseteq S$ is the set of streams using line o.

The situation which causes the largest buffer-need is when both all streams in S_o generate as much traffic as possible in as short time as possible, and when the streams not in S_o are idle, i.e. they do not delay the traffic of S_o.

In the worst case scenario, let t represent the time, and set $t = 0$ at the time traffic from the streams in S_o starts to get queued in the output buffer. If o is the connected to a source node we can define an upper bound on the number of cells arrived to the queue for line o in the time interval $[0, t]$ as

$$Arrived^{Src}(o,t) = \sum_{j \in S_o} \left(\left\lfloor \frac{t + J_{(j,0)}}{T_j} \right\rfloor + 1 \right) * N_j$$

For out-bound lines we can further bound the arrival of cells by exploiting the fact that the arrival rate is limited by the bandwidth of each in-bound line i (i.e. at most one cell every $tTrans_i$ will arrive from each in-bound line i). We obtain

$$Arrived^{Sw}(o,t) = \sum_{i \in \text{in-bound lines}} Arrived^{Sw}(i,o,t)$$

$$Arrived^{Sw}(i,o,t) = min \left(\underbrace{\left\lfloor \frac{t}{tTrans_i} \right\rfloor + 1}_{\text{Max delivered cells}}, \underbrace{\sum_{j \in S_o \cap S_i} G_j}_{\text{Generated cells}} \right)$$

$$G_j = \underbrace{W_{(j,h)}(t) * N_j}_{\text{Whole messages}} + \underbrace{min \left(\left\lfloor \frac{t - W_{(j,h)}(t) * T_j + J_{(j,1)}}{t_j} \right\rfloor + 1, N_j \right)}_{\text{A partial message}}$$

$Arrived^{Sw}(o,t)$ summarises the cells arrived from each in-bound line. $Arrived^{Sw}(i,o,t)$ represents (an upper bound on) the number of cells arrived from in-bound line i destined for out-bound line o at time t. Also, $S_o \cap S_i$ denotes the subset of S using both line o and line i. G_j is the number of cells generated in stream j at time t. Since line o can only be the output of either a source or the switch we can omit the ^{Src} and ^{Sw} superscripts.

For the considered worst case scenario we let $idle_o$ denote the first positive point in time when line o is idle. Then for $t \in [0, idle_o]$

$$Removed(o,t) = \left\lceil \frac{t}{tTrans_o} \right\rceil$$

represents the number of cells that have been removed from the queue in the time interval $[0, t]$

This yields the following equation for the buffer-need at time t $(0 \leq t \leq idle_o)$

$$Need(o,t) = Arrived(o,t) - Removed(o,t)$$

Note that at time 0 traffic has arrived ($Arrived(o,0) > 0$) while no cells have been removed ($Removed(o,0) = 0$). Thus, $Need(o,t)$ reflects that a cell has to pass through the queue, and use one buffer place, even if the cell is immediately dequeued.

In our search for the maximum buffer-need we do not need to consider times $t > idle_o$, since when the line goes idle any new burst of cells cannot require more buffers than the first burst (which captured the worst case scenario).

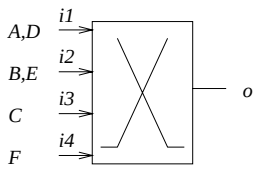
An algebraic solution to the maximum value of $Need(o,t)$ is not possible to construct, since the derivative of floor and ceiling operators are not meaningful. Instead, we note that both functions $Arrived(o,t)$ and $Removed(o,t)$ are monotonically increasing and that the function $Removed(o,t)$ increases only immediately after $\frac{t}{tTrans_o}$ is an integer. Thus, the maximum of $Need(o,t)$ must be exactly when $\frac{t}{tTrans_o}$ is an integer. Hence, to find the maximum $Need(o,t)$ it is sufficient to investigate times in $[0, idle_o]$ when this occurs.

As an example, consider the network and set of streams arriving to the switch described in figure 4. All streams have a message size $N_i = 1$ and all lines have $tTrans_l = 1$. In this example the deadlines, D_i, and propagation times, $tProp_l$, are irrelevant.

In this case $\frac{t}{tTrans_o}$ is an integer when t is an integer, so we should look for the maximum buffer-need using integral ts. Figure 5 shows how $Need(o,t)$ varies over time. In this example the load on the out-bound line is 93%.

Figure 5 indicate that even though the load is less than 100% it might take a long time before the buffer-need reaches 0 (i.e. $idle_o$ may be very large). Since $idle_o$ may be very large we will use an additional bound to limit the search for the maximum buffer need.

We know from scheduling theory (e.g. [2]) that a set of periodic processes have a cyclic pattern of execution. The length of this cycle is equal to the least common multiple (LCM) of the periods of the processes. Due to the jitter the cyclic behaviour will (in the worst case) be preceded by an initial stabilisation period.



Stream	T_i	$J_{(i,1)}$	P_i	$L_{(i,1)}$	$L_{(i,2)}$
A	2	1	5	i_1	o
B	7	2	4	i_2	o
C	7	2	3	i_3	o
D	14	4	2	i_1	o
E	14	4	1	i_4	o

Figure 4: Network and streams in buffer-need example

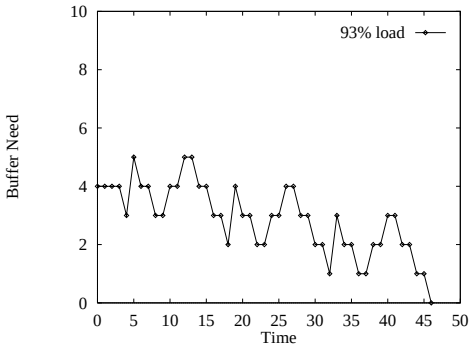


Figure 5: The buffer need over time

Each in-bound line has an individual stabilisation period which is dependent on the jitter of the streams using that line. The stabilisation period of an in-bound line l continues at most until the first idle-point of line l, this point can be expressed by the equation

$$init_l = \sum_{i \in S_l} (W_{(i,2)}(init_l) * N_i + min\left(\left\lceil \frac{init_l + J_i - W_{(i,2)}(init_l) * T_i}{t_i} \right\rceil, N_i\right)) * tTrans_l$$

When all in-bound lines have passed their stabilisation period the cyclic behaviour of cell arrivals to port o will start. Thus, the end of the first cycle can be defined as

$$cycle_o = \max_{l \in \text{in-bound lines}} init_l + LCM(\{T_i \mid i \in S_o\})$$

Combining $idle_o$ and $cycle_o$ we get an upper bound on the interval during which the maximum buffer-need occurs as

$$WCBound(o) = min(idle_o, cycle_o)$$

In the example above $cycle_o$ is 17 and $idle_o$ is as high as 46 which means that $WCBound(o) = min(46, 17)$. We can now define the maximum buffer-need as

$$MaxNeed(o) = \max_{0 \le t \le WCBound(o)} Need(o,t)$$

Remember that, $Need(o,t)$ is only defined in the interval $[0, idle_o]$. This means that we must calculate $idle_o$ if it is less than $cycle_o$. Since we know that $idle_o$ is the first point in time where $\frac{t}{tTrans_o}$ is an integer and $Need(o,t) = 0$, we can calculate $MaxNeed(o)$ as follows

$$\begin{aligned} &\text{calculate} && Need(o, k * tTrans_o) \text{ for } k = 0, 1, 2, \ldots \\ &\text{until} && \begin{cases} Need(o, k * tTrans_o) = 0 \text{ or} \\ k * tTrans_o > cycle_o \end{cases} \\ &MaxNeed(o) = && \max_k \; Need(o, k * tTrans_o) \end{aligned}$$

6. Examples

To illustrate some properties of response-time and buffer-need analysis. We will in this section present some sample networks and traffic situations.

To facilitate evaluation, we have developed a tool for response-time and buffer-need analysis. The tool accepts descriptions of a network and a set of streams as input data, and produces as output:

- Message response-times,
- message jitter,
- cell response-times for each hop,
- cell jitter after each hop, and
- buffer-need for each source and switch output buffer.

The tool can also be augmented to report intermediate values in the form of graphs (as in the buffer-need example in figure 5). The tool is implemented in the language Erlang [1].

6.1. A Simple Example

In this example we have 15 identical streams and a switch with only one input (i) and one output (o) line. The streams are defined in figure 6.

Stream	T_i	$J_{(i,0)}$	N_i	P_i	$L_{(i,1)}$	$L_{(i,2)}$
1	256	0	16	1	i	o
2	256	0	16	2	i	o
⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	256	0	16	15	i	o

$tSwitch^{max}$	$= 1$	$tSwitch^{min}$	$= 1$
$tTrans_i$	$= 1$	$tProp_i$	$= 5$
$tTrans_o$	$= 1$	$tProp_o$	$= 5$

Figure 6: Network and streams in the simple example

Each stream uses $\frac{1}{16}$-th of the bandwidth. This means the the utilisation of each line is $\frac{15}{16} \approx 94\%$.

In the stream definition in figure 7 we omitted the leaky bucket rate t_i. Meaningful t_is in this example are in the interval [1,15], where $t_i = 1$ means send each cell in one message back-to-back and $t_i = 15$ means spread the cells evenly over the whole period.

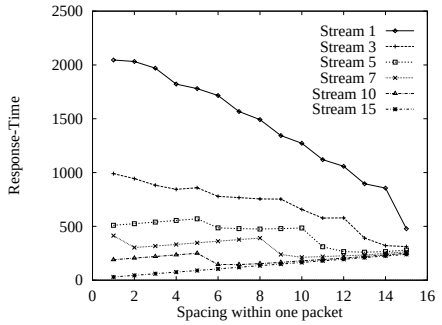


Figure 7: Response-times in the simple example

Figure 7 shows the response-times for streams 1, 3, 5, 7, 10 and 15, with t_i varying from 1 to 15. We make the following observations:

- Increased spacing radically decreases the response-time for the lowest priority stream. The reason for this is that spacing forces high-priority streams to leave empty slots that can be used by cells in the lower-priority streams.
- Spacing increases response-time for the high-priority streams, since it increases the time to pass through the leaky bucket.
- When spacing increases, the response-time for mid-priority streams will approach the response-time for high-priority streams, since the mid-priority streams will use the empty slots made available by the spacing of high-priority streams.
- Increased spacing will not decrease response-times for all streams. Even low-priority streams (e.g. stream 5 in figure 7) may experience increased response-time with increased spacing. The reason for this is that if the spacing is not increased enough to let more cells from the low-priority stream "slip in" between the spaced cells, then the low-priority stream will suffer from the increased time to send the spaced message, while not being able to benefit from the spacing of higher-priority streams.

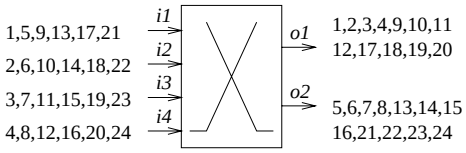
Buffer-need analysis gives for this example: $MaxNeed(i) = 240$ and $MaxNeed(o) = 1$ regardless of the spacing, t_i. From the definition of $Arrived^{Src}(o,t)$ it is obvious that the spacing does not affect the buffer-need at a source node (since we assume that the whole message is queued at once), and that the source node must be able to queue one whole message from each stream, i.e. 15*16 = 240 cells. As for $MaxNeed(o)$ it may seem more surprising that the buffer need is exactly 1 regardless of the spacing. The explanation is quite natural: Since all streams arrive on the same line and the out-bound line has the same capacity as the in-bound line, cells will arrive at the same rate as they are removed; yielding a buffer-need of 1.

It should be noted that the response-time analysis does not capture this *pipeline* behaviour. The response-time analysis includes interference from higher priority streams in both the source and the switch, even though such interference does not occur in this special case.

The pipeline behaviour occurs when the bandwidth of the out-bound line is equal to or larger than the total bandwidth of the in-bound lines carrying traffic destined for the out-bound line. (Since in that case the out-bound line is able to transmit an incoming cell before it is possible for the next cell from the same line to arrive.) The response-time analysis can easily be extended to test if pipeline effects exists and if so use a modified $Q_{(i,h)}$ which captures the limited interference from other streams.

6.2. A Larger Example

In this example we have 24 streams which are identical except for the priority and the path through the network. The network consists of four sources and two destinations. The streams and the network are described in figure 8.



Stream	T_i	$J_{(i,0)}$	N_i	P_i	$L_{(i,1)}$	$L_{(i,2)}$
1	256	0	16	1	se fig	
2	256	0	16	2	se fig	
⋮	⋮	⋮	⋮	⋮	⋮	
24	256	0	16	24	se fig	

$tSwitch^{max}$	$= 1$	$tSwitch^{min}$	$= 1$
$tTrans_l$	$= 1$	$tProp_l$	$= 5$

For each line l in $\{i1, i2, i3, i4, o1, o2\}$

Figure 8: Network and streams in the larger example

Again, each stream incurs a load of $\frac{1}{16}$-th (on both in-bound and out-bound lines). This means that we have a load of $\frac{6}{16}$ (37.5%) on the in-bound lines and $\frac{12}{16}$ (75%) on the out-bound lines.

As in the simple example, we calculate response-times and buffer-need for different spacing of messages,

using integral t_is in the interval [1,15].

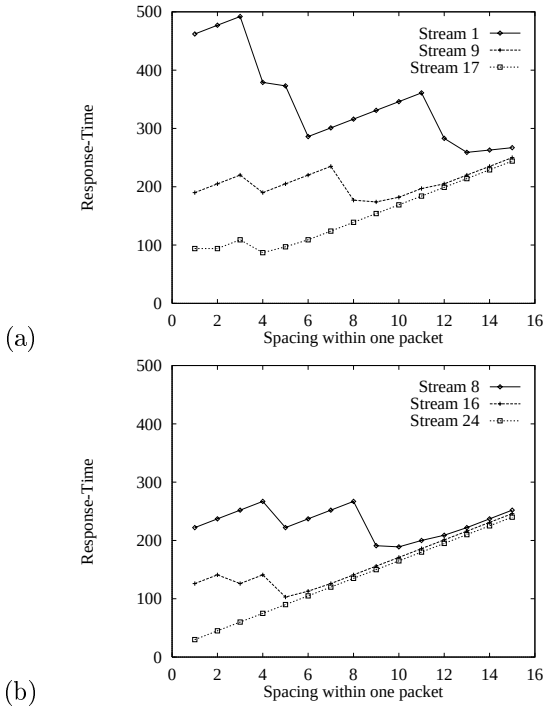


Figure 9: Response-times for streams on the same path

Figure 9 shows the response-times of the streams on two of the eight paths through the network. (In figure 9(a) for path $\{i1, o1\}$ and in figure 9(b) for path $\{i4, o2\}$.) Notable is that even the lowest priority stream, reaches a point where increased spacing of higher priority streams do not decrease the response-time. This is due to the relatively low load (75%) on the out-bound line. When the load is low the empty slots left by high-priority streams are not all used by mid-priority streams.

Figure 10 shows the response-times of the four lowest (a) and the four highest (b) priority streams. All streams in figure 10(b) uses out-bound line $o2$. This explains the (small) interference experienced by Stream 23 when $t_i = 1$. If Stream 24 and 23 would have used different in- and out-bound lines then their response times would have been exactly the same, since interference is only caused by higher priority streams for shared links.

The result from the buffer-need analysis is presented in figure 11. One curve is sufficient to describe all source buffer needs, since the traffic condition is the same on each source node. Also for the switch buffer-need we only plot one curve since the buffer-need is almost identical for the two switch queues. (The buffer-need differ by at most one cell, due to a difference in

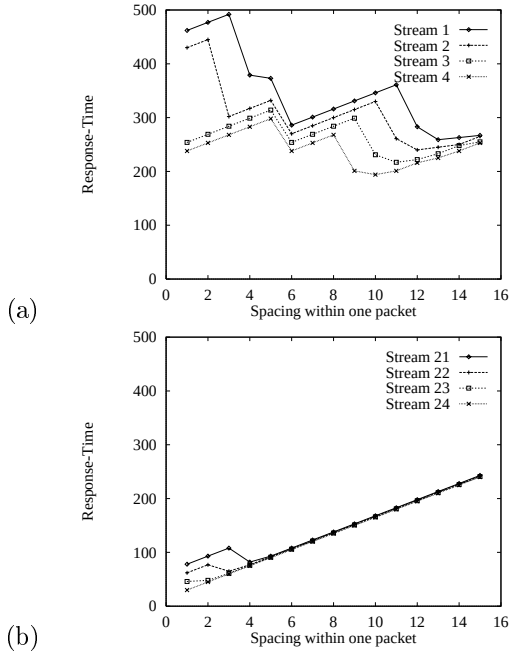


Figure 10: Response-times for the (a) lowest and (b) highest priority streams

jitter when the streams reach the switch.)

We note that there is a range in which increased spacing has a near linear positive impact on the buffer need. However,

- if spacing is too small, a small increase in spacing (e.g. from 1 to 2) will not decrease the rate at which buffers fill up in the switch, and
- when spacing is large enough (in this example 12), the spacing is enough to allow the buffers to start to drain even between individual cells in the messages, thus increasing spacing further will not have a major impact on the buffer-need. (The slow decrease in buffer need for spacing larger than 12 is solely due to the still decreasing jitter for low priority streams.)

7. Discussion

Our method for determining the response-time for messages are based on Fixed Priority Scheduling (FPS) of real-time processes, originally described by Liu and Layland [12]. The formulation of response-time equations for a FPS system was first made by Joseph and Pandya [9], and has been further developed at the university of York [3]. In particular, response-time equations have been derived for distributed real-time systems based on the *Controller Area Network* (CAN) [20] and hardware entities such as disk drives [18].

Various schemes to provide hard real-time guarantees for ATM connections have been proposed. Scheduling

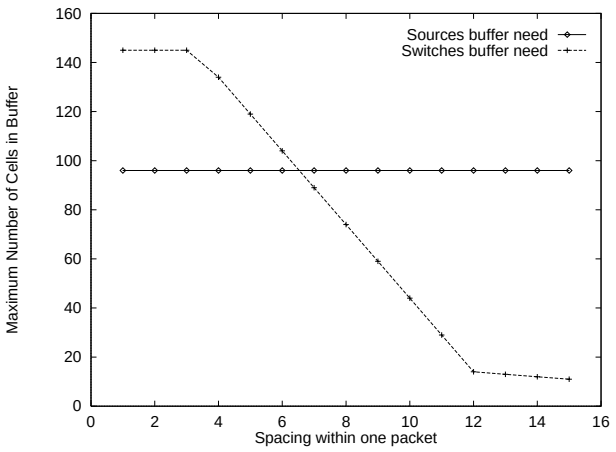


Figure 11: Buffer-need in the sources and the switch

policies are often classified as *work conserving* or *non-work conserving*. A switch using a work conserving policy is idle only if there are no cells queued in the switch, whereas a non-work conserving policy may introduce artificial delays. Work conserving schemes have lower average queueing-delay and buffer-need compared to non-work conserving schemes. Also, all work conserving schemes have equivalent average queueing-delay and maximum buffer-need. Our scheduling policy (priority queueing) is work conserving.

In general, a non-work conserving scheduling policy is more complex (in terms of implementation) than a work conserving policy, since the non-work conserving policy has to keep track of when cells can be transmitted, not only the order of transmission.

Two non-work conserving scheduling policies which have been proposed are Golestani's "Stop and Go Queueing" (SGQ) [7] and the Karlsson method [10]. SGQ does not allow fast delivery of urgent messages unless the complexity of the implementation is increased and multiple queues for each output port are implemented. The Karlsson method allows different streams to have different requirements on response-times, but due to its non-work conserving nature, urgent messages may be (unnecessarily) delayed.

Others have been able to provide deterministic guarantees with work conserving schemes, e.g. Raha, Malcolm and Zhao's "Weighted Round Robin" (WRR) [16] and Parekh and Gallager's "Packetised General Processor Sharing" (PGPS) [13, 14] (also known as "Weighted Fair Queueing" (WFQ) from [6]). WRR requires one separate queue for each stream. The queue management is quite complex since it is required to support a variable number of queues of various sizes. PGPS can be implemented with a "virtual clock" mechanism which eliminates the need for a separate queue for each stream. However, the output queue must be kept ordered by time stamps, which is quite expensive.

In the analysis presented in section 4 we assume unique priorities for each stream. Apart from the problem of assigning unique priorities in a large network, unique priorities for each stream will give similar problems as PGPS when maintaining the output queues. However, the analysis can be extended to allow arbitrary many streams to share the same priority. Yet unpublished experiments (performed by Ken Tindell), indicates that as few as 16 priority levels is sufficient to achieve practically the same response-times as with an infinite number of priority levels. Priority queues with as few as 16 or 256 priority levels can be implemented efficiently enough to support the bandwidth of ATM networks [5].

In the case of PGPS it is possible to decrease the response-time by over-allocating bandwidth. Obviously, over-allocation is not preferable, since it limits the total network utilisation. With our method, over-allocation in terms of larger N_is or smaller T_is is directly penalising since it will increase the response-time.

One of the major advantages of using priority queueing is the ability to give urgent messages short worst case response-times. In a typical real-time system there are a number of concurrent activities, with varying response-time requirements. A large portion of the traffic is not very urgent, while some messages may be part of tight control loops with very short deadlines. The examples in section 6 clearly shows that priority queueing makes it possible to accommodate streams with varying response-times requirements.

We can also learn from the examples in section 6 that by tuning priorities and spacing we can (to some extent) control the response-times. This is very useful, since if we can tune response-times close to the deadlines it will be possible to fit more streams into the system and thereby increase the network utilisation.

In this paper we have not treated priority and spacing assignment, but we know that for some non-distributed systems there exist optimal priority assignment algorithms, e.g. deadline monotonic priority assignment [11], and the more general approach in [2]. However, for distributed systems there exists no known optimal algorithm. Furthermore, we not only have to set priorities we also have to choose the cell spacing. This further increases the complexity of finding an optimal combination of priority assignments and cell spacings. Hence, it is likely that we have to rely on heuristic methods for priority and spacing assignment.

We have in this paper only treated hard real-time traffic, but it is possible to incorporate other QoS

classes into our scheme by reserving a low priority level for the non real-time traffic and having a separate pool of buffer memory for those streams.

References

[1] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in Erlang.* Prentice Hall, 2nd edition, 1996.

[2] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS-164, Dept. of Computer Science, University of York, England, November 1991. Available at ftp://ftp.cs.york.ac.uk/pub/realtime/papers/-YCS164.ps.Z.

[3] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell, and A.J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2/3):129–154, 1995.

[4] N.C. Audsley, A. Burns, K.W. Tindell, M.F. Richardson, and A.J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[5] H.J. Chao and N. Uzun. A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager. *IEEE Journal of Solid-State Circuits*, 27(11):1634–1643, November 1992.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *Internetwork: Research and Experience*, 1(1):3–26, September 1990.

[7] S. Jamaloddin Golestani. A stop-and-go queueing framework for congestion management. In *ACM SIGCOMM '90*, pages 8–18, September 1990.

[8] R. Händel, M.N. Huber, and S. Schröder. *ATM Networks Concepts, Protocols, Applications.* Addison-Wesley, second edition, 1994.

[9] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[10] G. Karlsson. Capacity Reservation in ATM Networks. *Computer Communications, Special issue on Algorithms for ATM Networks*, March 1996.

[11] J.Y.T. Leung and J. Whitehead. On the complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[12] C. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[13] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[14] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, April 1994.

[15] C. Partridge. *Gigabit Networking.* Computing Series. Addison-Wesley, 1994. ISBN 0-201-56333-9.

[16] A. Raha, N. Malcom, and W. Zhao. Hard real-time communications with weighted round robin service in ATM local area networks. In *Engineering of Complex Computer Systems*, volume 1, pages 96–103. IEEE, IEEE Computer Society Press, November 1995.

[17] M. Sjödin. Response-Time Analysis for ATM Networks. Technical Report DoCS 96/76, Department of Computer Systems, Uppsala University, 1996.

[18] K.W. Tindell and A. Burns. Fixed priority scheduling of hard real-time multimedia disk traffic. *The Computer Journal*, 37(8):691–697, 1994.

[19] K.W. Tindell and J. Clark. Holistsic Schedulability Analysis For Distributed Hard Real-Time Systems. Technical Report YCS197, Real-Time Systems Research Group, Department of Computer Science, University of York, November 1994.

[20] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In F. Jahanian and K. Ramamritham, editors, *Proc. 15th Real-Time Systems Symposium*, pages 259–263. IEEE Computer Society Press, 1994.