Grid-based switch fabrics: a new approach in designing fault-tolerant ATM switches

H.S. Laskaridis\textsuperscript{a,}\textsuperscript{*}, A.A. Veglis\textsuperscript{b}, G.I. Papadimitriou\textsuperscript{b}, A.S. Pombortsis\textsuperscript{a}

\textsuperscript{a}Department of Informatics, Aristotle University of Thessaloniki, P.O. Box 888, 54006 GR, Thessaloniki, Greece
\textsuperscript{b}Computer Lab, School of Journalism and Mass Communication, Aristotle University of Thessaloniki, 54006 GR, Thessaloniki, Greece

Received 27 January 2000; revised 10 January 2001; accepted 11 January 2001

Abstract

ATM is the switching and multiplexing technology chosen to be used in the implementation of B-ISDN, because of its superiority in fast packet switching. However, the use of ATM switches with large number of input and output ports have been proven to be a bottleneck in wide area ATM networks. In this paper, we propose a new space-division grid-based ATM architecture with fault tolerant characteristics and minimal number of switching elements (SE’s). © 2001 Elsevier Science B.V. All rights reserved.

Keywords: ATM switch fabric; Grid-based architecture; Fault-tolerance; Survival probability

1. Introduction

ATM is the switching and multiplexing technology chosen to be used in the implementation of B-ISDN, because of its superiority in fast packet switching. The use of ATM in wide area networks has revealed the necessity of ATM switches with large number of input and output ports. However, it has become obvious that there is a bottleneck in wide area ATM networks, which is due to medium’s bandwidth (fibre optic with extremely high bandwidth) but due to switches’ throughput. More precisely, the bottleneck is located in the ‘cell switch fabric’, which is the core of the ATM switch [1]. A lot of research effort is now focused on switch fabric architectures in order to increase the throughput of the switch fabric, while keeping the complexity at a reasonable level.

There are basically four categories of ATM switch fabric architectures [1]:

- \textit{Shared-medium architectures}: incoming cells are transferred to output ports through a shared bus (most commonly, e.g. [2]) or ring. A ‘shared-medium arbitration scheme’ controls the access of input ports to the shared medium.
- \textit{Shared-memory architectures}: incoming cells are written in a dual-port random access memory and read out of the memory by the output ports. The memory management and the input–output scheduling is performed by a controller.
- \textit{Fully interconnected architectures}: an independent path is available between each pair of input and output ports through the switch fabric. The crossbar switch is the most common architecture of this category. Variations of crossbar switches can be found in Refs. [3,4].
- \textit{Space-division architectures} or multistage interconnection networks (MIN): a path is available between each pair of input and output ports through a number of shared links and switching elements (SE’s).

The disadvantages of all proposed architectures belonging to the first three categories become obvious when we attempt to scale to switches with large number of input and output ports (e.g. 512 × 512). Firstly, throughput is found to be poor and secondly the hardware requirements are sometimes just not applicable (e.g. shared-memory with extremely high transfer rate).

For the above-mentioned reasons, a great research effort is converged on space-division architectures. The majority of them are based on Banyan MIN (e.g. [5–10]), and the aim is to improve its throughput and alleviate its major disadvantage, being internally blocking, by using various techniques. Another technique is building large-scale modular switches by interconnecting smaller switch fabrics. The interconnection network can be of various architectures. Such architectures are presented in Refs. [11–15] (although some of these papers are mainly focused on the multicasting property).
However, none of the well-known products uses space-
division architectures. Surprisingly, industry currently uses
architectures based on the shared-memory and shared-
medium (bus) models in the large majority of the products,
while there are only a few high-end switches using fully
interconnected architectures (‘crossbars’) [16,17]. The
throughput of Cisco’s catalyst family products ranges
from 3.6–32 Gbps. Marconi’s ASX family products [18]
are equipped with shared-memory switch fabrics, with
throughput ranging between 2.5 and 10 Gbps. Such through-
put is inadequate considering that line rate of 2.4 Gbps is
now becoming common, especially for Internet Service
Providers.

In this paper, we propose a new space-division switch
architecture, called Grid-based ATM Switch Architecture
(GASA). This architecture is not based on Banyan
networks, and its topology is grid-based. The grid is formed
by SE’s. One of the main characteristics of the proposed
architecture is the number of SE’s, which is equal to the
number of input and the number of output ports. Taking
under consideration the fact that central routing control in
a large ATM switch is usually proved to be another bottle-
neck, GASA is designed to be self-routing.

Fault tolerance is another issue in ATM switches’ design.
The majority of the fault tolerant architectures employ
redundant SE’s and/or links (e.g. [19–22]). Such architec-
tures are usually referred to as ‘dilated’ interconnection
networks and can exhibit fault tolerant characteristics, if
the proper logic is embedded. However, fault tolerance is
not described in these papers. In Refs. [23–25], fault toler-
ant switch architectures, also employing redundancy, were
presented and analysed. GASA also shows an excellent
behaviour regarding fault-tolerance, caused by the existence
of multiple paths between any input—output pair. The neces-
sary modifications in the switch basic architecture, in order
to support fault-tolerant routing, are studied along with its
robustness.

The rest of the paper is organised as follows: in Section 2
the overall architecture is presented in detail, including the
routing algorithm, executed in each switching element. In
Section 3 we present the shared-memory architecture of
each switching element, which can easily support any
kind of priorities. The analytical model of the switch archite-
cture is studied in Section 4, as well as simulation results
(cell loss probability and delay). For comparison reasons,
switch architecture with similar topology but quite different
operation, based on Manhattan Street Network (MSN), was
also simulated. Fault tolerant enhanced architecture is
presented in Section 5, and analysed and compared to
other fault tolerant switch architectures in Section 6.
Finally, concluding remarks can be found in Section 7.

2. The overall architecture

The main characteristic of GASA is the number of the
necessary SE’s, which is minimal, comparing with other
switching architectures. More specifically it is equal to the
number of input ports and the number of output ports. Each
SE is directly connected to an input module, an output
module and its 2–4 (depending on its position on the grid)
neighbour SE’s. Links between SE’s are bi-directional
while links between SE’s and input or output modules are
unidirectional. Internal links’ rate is equal to incoming and
outgoing links’ rate and no speed-up factor is deployed in
the basic implementation. A 16 × 16 switch is shown in
Fig. 1. The number of SE’s, the number of unidirectional
links (a bi-directional link is equivalent to two unidirec-
tional links) and the number of hops are presented in Tables
1–3 respectively, in comparison to those of a Banyan
network.

Each SE has an address, a Switching Element ID (SE ID)
which is used in routing, as explained in Section 2.1. The
SE ID is a binary number whose length (in bits) depends on
the size of the switch. Addresses are assigned in a recursive
way, as shown in Fig. 2, in a way similar to the one used
in the hyper-cube architecture (although the SE’s are not
linked in the same way).

It is also worth noting that, for reasons of homogeneous
implementation of the routing algorithm, a 2-bit prefix is
added when expanding from N × N square switch to
2N × 2N non-square switch (e.g. from 16 × 16 to 32 × 32),
although a single-bit prefix would be adequate for having
each SE uniquely identified.

2.1. Routing algorithm

GASA is a self-routing architecture. Each SE is capable
of routing the incoming cells towards their destination,
using only the destination SE ID. When a new cell arrives,
the corresponding input module searches for the source VPI/
VCI and port number in a lookup table, in order to find the
destination output port and the corresponding destination
SE ID. The destination SE ID is placed in the tag (header)
of the cell used internally in the switch (refer to Section 3 for
a complete description of the cell tag).

The following routing algorithm for a 16 × 16 switch is
executed in each SE independently of its position on the
grid. In this algorithm, SE ID[a] denotes the ath bit of
the SE ID of the current SE, SE ID[a,b] denotes the ath
and bth bits of the SE ID, and Dest SE ID[a] denotes
the ath bit of the SE ID of the destination SE.

If Dest SE ID[1,2] ( ) SE ID[1,2] then
Route on (1,2)
Else If Dest SE ID[3,4] ( ) SE ID[3,4] then
Route on (3,4)
Else
Send to output module
Procedure Route on (a,b)
If Dest SE ID[a] ( ) SE ID[a] then
If Dest_SE_ID[a] = 0 then
    Send to North
Else {Dest_SE_ID[a] = 1}
    Send to South
Else {Dest_SE_ID[a] = SE_ID[a], but}
    {Dest_SE_ID[b]}(SE_ID[b])
If Dest_SE_ID[b] = 0 then
    Send to West
Else
    Send to East

Each cell is initially routed to the proper ‘quadrant’ of the grid, based on the first two bits of the destination SE ID, then moved to the proper SE, based on the last two bits, and then routed to the corresponding output module.

The routing algorithm exhibits the following characteristics:

- all cells follow shortest paths,
- all cells in general do not follow paths of the same length, but this is not of major importance because
- all cells from SE A to SE B follow the same path, which justifies the absence of reassembly buffers.

The fact that not all SE’s have neighbours in all directions does not affect the correctness of the above routing algorithm. For example, in a SE that does not have north neighbour the corresponding command (“Send to North”) is not going to be executed ever, as the corresponding condition is not going to be fulfilled.

Next we demonstrate the expandability of the routing algorithm. The algorithm for a 32×32 or a 64×64 switch is presented:

If Dest_SE_ID[1, 2](SE_ID[1, 2]) then
    Route on (1, 2)
Else If Dest_SE_ID[3, 4](SE_ID[3, 4]) then
    Route on (3, 4)
Else If Dest_SE_ID[5, 6](SE_ID[5, 6]) then
    Route on (5, 6)
Else
    Send to output module
(The Route_on (a, b) procedure remains the same.)

The disadvantage of the architecture is that it is internally blocking. Two to four cells may require to be transmitted simultaneously from a SE over the same outgoing link. This problem can be solved by using an internal shared buffer in each SE. MSN, which have similar topology but different routing algorithm, employ deflection routing to overcome this problem ([26,27]). The drawback of deflection routing is that cells from a source to a destination do not always follow the same path. Different paths that cells may follow, may have different lengths, which makes the use of reassembly buffers mandatory. In this way hardware complexity increases. This is the reason why a deterministic algorithm was preferred in GASA.

3. Switching element architecture

Although the number of SE’s in GASA is significantly smaller than the corresponding in Banyan networks, the hardware complexity is quite higher. There are SE’s with

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Number of SE’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x16</td>
<td>32×32</td>
</tr>
<tr>
<td>Banyan $(N/2)\log_2 N$</td>
<td>32</td>
</tr>
<tr>
<td>GASA $N$</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Number of links</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16x16</td>
</tr>
<tr>
<td>Banyan $(N\log_2 N+1)$</td>
<td>80</td>
</tr>
<tr>
<td>GASA $6N - 4\sqrt{N}$ (square grid)</td>
<td>80</td>
</tr>
<tr>
<td>or $6N - 6\sqrt{2}$ (non-square grid)</td>
<td></td>
</tr>
</tbody>
</table>
2, 3 or 4 neighbour SE's. However, the deployed architecture should be similar in all SE's, independently of the number of inputs/outputs ports. For this reason, shared-memory architecture is deployed. Specifically, in Ref. [28] Hashemi and Leon-Garcia propose a shared-buffer switch architecture, a variation of which is deployed in our switch.

The disadvantage of using a shared-buffer architecture for an $n \times n$ switch is that the shared buffer has to operate at a rate $n$ time higher than each link rate. In other words the throughput of the shared buffer has to be $n$ times greater than each link rate. However, this is not a major problem in our switch, because $n$ is small ($3 \leq n \leq 5$).

On the other hand the advantage of using a shared buffer architecture in contrast to using separate queues (one for each output port) is that we have better memory utilisation. In other words, we need less memory in order to obtain the same cell loss probability. Next the operation of the shared queue is described.

Each queue consists of groups of cells. Each group consists of cells that will be transmitted over different outgoing links during the same timeslot. Cells entering the SE enter the queue from the head of the queue and start searching for their position (Fig. 3). Each cell is placed at the end of the first group that does not have a cell for the same outgoing link and makes all cells buffered in the following slots to move one slot backwards.

If priorities are used, then each incoming cell is not simply looking for the first group with no other cell trying to use the same outgoing link, but its 'priority control' field (Fig. 4) is also compared to the corresponding field of cells already buffered that will use the same outgoing link. If the new cell has higher priority than the buffered cell, the new cell replaces the old one, which starts searching for a new position moving backwards. Otherwise, the new cell keeps searching its position in the buffer.

In order for the above scheme to operate, the use of a tag is necessary. The internal structure of a cell is shown in Fig. 4. The cell tag is inserted by the input module, when the cell enters the switch. It has 4 fields. The values of the first two fields, ‘destination SE ID’ and priority control, are assigned only once by the input modules. On the other hand, the values of the third and fourth field are updated by each SE and are used in the operation of the SE. The ‘output port ID’ field is set by the routing algorithm, based on the next SE the cell has to move to (north, south, west, east, or output module). The ‘end of group’ field is set by the queue control to delimit each group.

It is worth noting that there are several ways that the priority control field could be used. For example the value assigned by the input module could be based on the quality of service that is requested for the corresponding virtual connection, or it could be a function of the ‘age’ of the cell in the switch (which means that each SE should be able to update this field). The field could also be split into two sub-fields in order to combine the two priority control schemes. However, priority control schemes will not be
further discussed in this paper. In the following we assume that all cells are of equal priority.

4. Performance analysis

4.1. Analytical model

The analytical model is based on the following assumptions [29] concerning the operation of the switch:
1. Cells’ arrivals from input x to output y follow a Bernoulli distribution with rate \( r_{xy} \). No assumption about uniform distribution of incoming traffic to the outputs is made.
2. During a timeslot 0 to n cells arrive in each SE from its incoming links, and 0 to n cells depart from the SE over its outgoing links.
3. No ‘grant’ signals are exchanged between SE’s prior to cell transmissions. Arriving cells that cannot find space in the shared buffer are discarded.
4. The analytical model is a discrete time model, although technically there is nothing preventing the switch from operating asynchronously. Moreover we are going to assume that arrivals and departures of cells take place serially, in this order, and not simultaneously.

We note that ‘cells arriving at a queue’ are not necessarily stored in the queue. This expression is used to indicate that cells arrive at a SE trying to be enqueued, but they may be dropped if no space is available.

Most of the notations employed in the analysis can be found in Table 4.

In order to perform our analysis we employ an equivalent model for the SE: we consider each SE having a dedicated queue per output port (from now on called ‘virtual queue’). Virtual queue \( (x, d) \) is the virtual queue in SE \( x \) corresponding to the outgoing link in the d direction, where \( d \in \{N, S, W, E, O\} \) (i.e. north, south, west, east and output, respectively). However we do not pose any restrictions on the length of each virtual queue. We only pose restrictions on the summation of lengths of the queues belonging to the same switching element:

\[
\sigma = \sum_{d} s_{(x,d)} \leq B
\]

This way the model remains accurate.

According to the above-mentioned assumption 4, we define the following state and transition probabilities:

- \( \pi^{S}_{(x,d)}(s, \sigma) \), the probability that the virtual queue \( (x,d) \) is in state \((s, \sigma)\) after cell arrivals from neighbour SE’s have completed, but no cell departures have taken place.
- \( \pi^{P}_{(x,d)}(s, \sigma) \), the probability that the virtual queue \( (x,d) \) is in state \((s, \sigma)\) after cell departures have completed.
- \( \lambda_{(x,d)}^{S}(s_1, \sigma_1 | s_1, \sigma_1) \), the probability that the virtual queue \( (x,d) \) makes a transition from state \((s_1, \sigma_1)\) to state \((s_2, \sigma_2)\), after cell arrivals have completed, but no cell departures have taken place.
- \( \lambda_{(x,d)}^{P}(s_1, \sigma_1 | s_1, \sigma_1) \), the probability that the virtual queue \( (x,d) \) makes a transition from state \((s_1, \sigma_1)\) to state \((s_2, \sigma_2)\), after cell departures have completed.

The state equations for each virtual queue can be written as:

\[
\begin{align*}
\pi^{S}_{(x,d)}(s_2, \sigma_2) &= \sum_{s_1=0}^{\sigma_1} \sum_{\sigma_1=0}^{\min(s_1, \sigma_1)} \pi^{S}_{(x,d)}(s_1, \sigma_1) \cdot \lambda_{(x,d)}^{S}(s_2, \sigma_2 | s_1, \sigma_1), \\
\pi^{P}_{(x,d)}(s_2, \sigma_2) &= \sum_{\sigma_1=\sigma_2}^{\min(\sigma_2 + n_x)} \sum_{s_1=\sigma_2}^{\sigma_1} \pi^{P}_{(x,d)}(s_1, \sigma_1) \cdot \lambda_{(x,d)}^{P}(s_2, \sigma_2 | s_1, \sigma_1).
\end{align*}
\]

We define \( \text{Adj}(x) \) as the set of SE’s that are adjacent to \( SE_x \).

The input and output modules connected to the specific switching element are also considered as neighbours and included in \( \text{Adj}(x) \). We also define:

- \( \text{adj}(x,N) \): the north neighbour of \( SE_x \)
- \( \text{adj}(x,S) \): the south neighbour of \( SE_x \)
- \( \text{adj}(x,W) \): the west neighbour of \( SE_x \)
- \( \text{adj}(x,E) \): the east neighbour of \( SE_x \)

In the following \( SE_x \) represents the neighbour of \( SE_x \) in the d direction. So, cells stored in the virtual queue \( (x,d) \) are destined to \( SE_y \). In order to calculate the probability of \( k_{(x,d)} \) cells arriving in queue \( (x,d) \), we first define:

- \( p^{N}_{(x,d)} = \sum_{r_{xy}} \forall v, w \): the probability that \( k_{(x,d)} \) cells arriving in queue \( (x,d) \) from the north neighbour \( SE_x \).
- \( p^{S}_{(x,d)} = \sum_{r_{xy}} \forall v, w \): the probability that \( k_{(x,d)} \) cells arriving in queue \( (x,d) \) from the south neighbour \( SE_x \).
- \( p^{W}_{(x,d)}, p^{E}_{(x,d)} \) are defined in similar way, for the west and east neighbours.

![Fig. 3. Shared-buffer switching element architecture.](image)

![Fig. 4. Internal cell structure.](image)
where $|D|$ is the number of elements that subset $D$ has, and

$$M_{x,{\text{adj}(x,d)}} = \sum_{\sigma \geq x, \sigma > 0} \tau_{x,d}(s, \sigma).$$

So in order to calculate this probability, we consider all the combinations of $k$ inputs sending cells (set $D$ in Eq. (3)), while the rest of the inputs ($|\text{Adj}(x)| - k$) do not send cells (set $D$ in Eq. (3)). $k_{x,d}$ of the arriving cells are destined to the virtual queue $(x,d)$ (set $D1$ in Eq. (3)), while the rest ($k - k_{x,d}$) cells are destined to other virtual queues (set $D - D1$ in Eq. (3)).

Similarly, the ‘departure probability’, the probability that $m$ cells depart from $SE_x$ and $(x,d)$ cells depart from virtual queue $(x,d)$ during a timeslot is:

$$q_{x,d}(0, m) = \sum_{|D| = m, D \subseteq D} \left( \prod_{d' \in D} (1 - M_{x,{\text{adj}(x,d')}}) \right),$$

$$q_{x,d}(1, m) = \sum_{|D| = m, D \subseteq D} \left( \prod_{d' \in D} (1 - M_{x,{\text{adj}(x,d')}}) \right),$$

$$q_{x,d}(m_{x,d}, m) = 0, \text{ if } m_{x,d} > 1.$$  

In this case we consider all the combinations of $m$ output ports transmitting cells (set $D$ in Eq. (5)), while the rest of them ($|\text{Adj}(x)| - m$) do not transmit. Having calculated arrival and departure probabilities, we are now able to calculate transition probabilities for each queue $(x,d)$:

$$p_{x,d}(k_{x,d}, k) = \sum_{\forall D \subseteq \text{Adj}(x)} \sum_{|D| = k_{x,d}} \left( \prod_{d' \in D} p^D_{x,d}(d') \right) \prod_{d \notin D} (M_{x,{\text{adj}(x,d)}} (1 - P^D_{x,d}(d))).$$

$$\lambda_{x,d}(s_2, \sigma_2|s_1, \sigma_1) = \begin{cases} p_{x,d}(s_2 - s_1, \sigma_2 - \sigma_1), & \text{if } \sigma_2 < B \\ p_{x,d}(k_{x,d}, k), & \text{if } \sigma_2 = B \text{ and } \exists j \geq 0 : \sigma_1 + k = B + j, \end{cases}$$

where

$$a = \frac{P[\text{cell arrived at } vq(x,d)]}{P[\text{cell arrives at } SE_x]} = \frac{\sum_{k=1}^{n_x} \sum_{k_{x,d}=1}^{k} p_{x,d}(k_{x,d}, k)}{\sum_{k=1}^{n_x} \sum_{k_{x,d}=0}^{k} p_{x,d}(k_{x,d}, k)}.$$  

$$\lambda'_{x,d}(s_2, \sigma_2|s_1, \sigma_1) = q_{x,d}(s_1 - s_2, \sigma_1 - \sigma_2).$$

In Eq. (6) we assume that $k$ cells arrive at the $SE_x$, but
\[ \sigma_1 + k \geq B \Rightarrow k \geq B - \sigma_1. \] More precisely, \( \sigma_1 + k = B + f(j \geq 0) \). So we assume that \( j \) cells are dropped. \( [a:j] \) of these cells were destined to virtual queue \((x,d)\), where \( a \) is the part of cells arriving at SE \(_x\) that are destined to \((x,d)\). So:

\[ s_2 = s_1 + k_{(x,d)} - [a:j] \Rightarrow k_{(x,d)} = s_2 - s_1 + [a:j]. \]

The set of Eqs. (1)–(8) forms an open system, which can be solved by recursions, resulting in the set of steady state probabilities for each virtual queue.

**Throughput**

The throughput of virtual queue \((x,d)\) under the specific load matrix, is:

\[ t_{(x,d)} = 1 - \sum_{\sigma=0}^{B} \pi_{(x,d)}(0, \sigma). \]  

(9)

We define \( \pi_x(\sigma) = \sum_{s=0}^{\infty} \pi_{(x,d)}(s, \sigma) \) the probability of SE \(_x\) having \( \sigma \) cells in its shared buffer. (The summation is calculated over anyone of its virtual queues.) The total throughput of SE \(_x\) is:

\[ t_x = 1 - \pi_x(0), \]  

(10)

while the total throughput of the switch can be calculated over the virtual queues that correspond to output ports connected to output modules:

\[ t = \sum_{x=0}^{N-1} t_{(x,d)}. \]  

(11)

**Loss probability**

The cell loss probability in SE \(_x\) is:

\[ Lp_{x} = \sum_{\sigma=0}^{B} \pi_{x}(\sigma) \sum_{k_{(x,d)}=0}^{k} p_{(x,d)}(k_{(x,d)}, k). \]  

(12)

The pass probability of a cell moving from SE \(_x\) to SE \(_y\) is calculated over all SE’s belonging to the path from SE \(_x\) to SE \(_y\): \( \prod_{\forall z \in \text{path}(x,y)} (1 - Lp_{z}). \)

So the total cell loss probability of the switch under the specific traffic pattern is the ‘weighted’ average of loss probabilities of all paths:

\[ Lp = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \left( \frac{r_{xy}}{\rho} \left( 1 - \prod_{\forall z \in \text{path}(x,y)} (1 - Lp_{z}) \right) \right), \]  

(13)

where

\[ \rho = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} r_{xy}, \]  

(14)

is the total load of the switch.

**Delay**

In order to calculate the average delay of cells, we first calculate the average queue length, average path length and average throughput. The average length of SE \(_x\)’s shared queue is:

\[ \bar{\mu}_{x} = \sum_{\sigma=0}^{B} (\sigma \pi_{x}(\sigma)). \]  

(15)

and the average queue length of all SE’s is:

\[ \bar{\mu} = \frac{1}{N} \sum_{x=0}^{N-1} \bar{\mu}_{x}. \]  

(16)

The average path length is also calculated as a ‘weighted’ average, taking under consideration the traffic pattern (i.e. the load matrix):

\[ pl_{\text{avg}} = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \frac{r_{xy}}{\rho} pl_{xy}, \]  

(17)

where \( pl_{xy} \) is the length (i.e. number of hops) of the path from SE \(_x\) to SE \(_y\).

The average throughput is:

\[ \bar{t} = \frac{1}{N} \sum_{x=0}^{N-1} \bar{t}_{x}. \]  

(18)

Finally, average delay according to Little’s law is:

\[ \bar{t} = \frac{\bar{\mu} pl_{\text{avg}}}{\bar{t}} + pl_{\text{avg}} = \left( 1 + \frac{\bar{\mu}}{\bar{t}} \right) pl_{\text{avg}}, \]  

(19)

where the first factor is the queue delay and the second factor is the process delay: cells remain in each SE that they arrive for at least one timeslot, even if there are no cells pending in the shared queue. In other words a cell arriving at a SE at the current timeslot cannot leave the SE earlier than the beginning of the next timeslot.

**4.2. Simulation results**

In order to cross-check the correctness of the analytical model, a software tool simulating a 16 × 16 GASA switch was developed. Several simulations with different traffic loads were contacted. More specifically we employed 6 load matrices with total load 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9. The load from input port \( x \) is uniformly distributed to all output ports \( y \neq x \). On the other hand, as input \( x \) and output \( x \) are used to connect the same end-station to the ATM switch, it has to be \( r_{xx} = 0 \). Fig. 5 shows the total normalised throughput (i.e. \( t_{xy} = t/\rho \)) and cell loss probability as calculated by the analytical model and measured by simulation for several loads.

The analytical model curve follows closely the simulation curve. The difference between the results is insignificant, as the analytical model describes the operation of the switch in detail, without making any unrealistic assumptions.

Simulation results reveal that GASA does not behave properly for loads greater than 0.6. Great loss probability was exhibited, which could not be alleviated by increasing buffer capacity, as depicted in Fig. 6. The problem is due to central links overload (which is of course irrelevant to buffer
capacity). Under uniform traffic, the load is not uniformly distributed over the links and central links get saturated.

In order to overcome this problem we developed two other models: GASA-2 with its 4 central links operating at double speed and GASA-3 with all of its links operating at double speed compared to the input and output links.

Fig. 6 shows the cell loss probability versus the buffer capacity (in cell slots) for GASA-1 (i.e. the basic architecture, without any speed-up factor for internal links). Fig. 7 shows the improvement in cell loss probability exhibited in GASA-2 (which is a minor improvement) and GASA-3 (which is quite a major improvement), for total loads equal to 0.5 and 0.6. We note that the buffer capacity shown in all graphs is the capacity of each SE.

Figs. 8 and 9 show the cell loss probability and cell delay (in time slots), respectively, for GASA-3 and for the whole range of total load (0.5–0.9). As was expected, by increasing the buffer capacity, loss probability is drastically reduced, while delay is insignificantly (in low loads) or slightly (in high loads) increased. We can also note that when load increases both cell loss probability and delay increase, as some links get saturated.

Overall the GASA-3 architecture appears to have good performance under all loads without using very large buffers.

4.3. Manhattan Street Network simulation

For comparison reasons we created another software simulation tool, simulating an ATM switch fabric using the MSN topology (Fig. 10 — for simplicity the input and output modules as well as their adjacent links are omitted). Such a switch fabric would be of hardware complexity similar or higher (as explained later) compared to GASA switch.

As mentioned before, the use of deflection routing, which is employed in MSN, makes the use of reassembly buffers mandatory. The basic idea is to investigate and compare the performance of an MSN switch having in each SE a reassembly buffer of capacity equal to the capacity of the shared

Fig. 5. Analytical vs. simulation results: (a) normalised throughput; (b) cell loss probability.

Fig. 6. GASA-1 (basic architecture) loss probability, under various loads and buffer sizes.
buffer used in each SE in GASA switch, while no other buffer is used.

As shown in Fig. 10, each SE has 2 outgoing links towards neighbour SE’s, 2 incoming links from neighbour SE’s, plus an incoming and an outgoing link connecting the SE to the input and output modules respectively (which are not shown in Fig. 10). All the links are unidirectional.

The following rules are used in routing cells and in resolving blocking situations ([26,30]):

- A new cell trying to enter the switch, can only be forwarded from the SE-point-of-entrance, if and only if there are less than 2 other cells trying to use the outgoing links towards neighbour SE’s. If this is not the case, the new cell is dropped at the point of entrance.
- Each cell in each SE has a preference regarding the outgoing link that it is going to use for the next hop, in the next timeslot. The preference is computed by considering which is the shortest path to the destination. Preference can be assigned one of the values: row (i.e. the cell prefers to be transmitted over the ‘horizontal’ link), column (i.e. the cell prefers to be transmitted over the ‘vertical’ link), out-module (i.e. the cell prefers to be transmitted over the link connecting the SE to the corresponding output module), don’t-care (i.e. the cell does not have any preference whether it is transmitted over the horizontal or vertical link).
- If one of the incoming cells is destined to the output module connected to the SEc, then none of the cells is dropped in SEc, in any case.
- If two cells have different specific (i.e. not ‘don’t-care’) preferences, then their preferences are satisfied. If they have the same specific preference, then one of them is chosen in random and is satisfied while the other is deflected.
- If there are two incoming cells, one of them having a
specific preference, while the other has a don’t-care preference, then the former is satisfied while the latter uses the remaining link. If both incoming cells have don’t-care preference, then the one that came from the horizontal incoming link is transmitted over the horizontal outgoing link and the one that came from the vertical incoming link is transmitted over the vertical outgoing link.

- A cell destined to the output module connected to SEi would be buffered in the reassembly buffer of SEi if it is out of order (i.e. there are cells from the same source destined to SEi, that have not yet arrived). If the reassembly buffer is full the cell is dropped.

- In order to prevent cells from moving from SE to SE for too long or waiting in a reassembly buffer for too long, we impose a time limit (time-to-live, TTL). Cells that exceed the time limit are delivered, if they are in a reassembly buffer, or otherwise they are dropped.

We have run simulations for several values of B (reassembly buffer capacity) in the range of 5–30 cell slots, and for several values of time limit in the range of 8–32. The load matrices that were used for GASA simulations were also used in these simulations. Simulation results (shown in Figs. 11 and 12) prove that MSN architecture cannot operate as a switch fabric, unless some improvements are made. No correlation was proven between the reassembly buffer capacity and loss probability: loss probability remains constant, unaffected of buffer capacity increase.

Table 5 proves that the major problem of MSN architecture is that a large portion of the cells trying to enter the switch fabric are dropped at the point of entrance, as there is no outgoing link available nor any buffer for the cells to be temporarily stored. We note that the analysis of loss probability for the rest of the loads is similar. In order to alleviate the problem an input buffer is employed in each SE so that new cells arriving at the switch fabric not being able to be transmitted are temporarily stored in the SE’s input buffer, as suggested in Ref. [31].

In order to find the best combination of the parameters (TTL and input buffer capacity) we run several simulations for a variety of values of these parameters. The results are shown in Figs. 13 and 14.

When increasing the input buffer capacity without increasing TTL, the majority of losses are due to cells exceeding the TTL. So an increase in TTL is mandatory. However, no fine-tune could be found that would result in acceptable results, regarding loss probability and delay. Our results are compliant with the results presented in Ref. [31], although the metric measured in Ref. [31] is throughput.

We should also note that while creating the simulation tool, we found that the implementation of the above-mentioned rules regarding routing and contention resolution is a complex task, leading to an architecture with increased complexity in comparison to the GASA architecture. Concluding this section we would like to emphasise the difference on performance between the two architectures, in contrast to their resemblance regarding their topology.

![Manhattan street network](image_url)
5. Fault tolerant enhanced architecture

In order to enable fault tolerant capabilities, certain modifications have to be made:

1. There should be a central unit controlling and resolving faults, from now on called ‘Fault Recovery Control Unit’ (FRCU).
2. There should be a ‘communication network’ connecting each SE to the FRCU. A number of alternatives will be presented, along with their advantages and disadvantages.
3. There should be hardware in each port of each SE able to identify and report faulty links.
4. The routing algorithm should be adjusted, in order to be able to operate under a different way, when FRCU announces a fault.

| Table 5 |
|-----------------|-----------------|-----------------|
|                | TTL = 8         | TTL = 12        | TTL = 16        |
| On entering the grid | 74.53           | 87.35           | 93.01           |
| Due to time limit | 25.46           | 12.64           | 6.98            |
| Due to full reassembly buffers | 0              | 0               | 0               |

5.1. The fault recovery communication network

The ‘fault recovery subsystem’ consists of the ‘Fault Recovery Control Unit’ (FRCU) and the communication network connecting FRCU to each SE and vice versa. The operation of FRCU is presented in Section 5.2, while in the present subsection the alternatives that can be deployed for the communication network are presented.

The possible alternatives (shown in Fig. 15) are the following:
1. Use of a shared bus. All SE’s as well as the FRCU are connected to a shared bus. The advantage of this approach is low cost, while on the other hand the shared bus is a ‘single point of failure’.
2. Use of dedicated lines between each SE and the FRCU. Each SE is directly connected to the FRCU by a dedicated bi-directional link. This approach provides maximum fault tolerance, while its disadvantages are high cost and restricted scalability, due to the fact that the FRCU should have as many ports as the number of SE’s in the switch.
3. Use of multiple buses. Each row of SE’s has its own bus. FRCU is connected to all buses. This solution is considered to be a compromise between the above-mentioned approaches.

The bandwidth of the buses used in alternatives 1 and 3
are not of major importance, as the traffic on these buses is minimal.

It is worth noting that failure in any component of the communication network - links, buses or even the FRCU - results in fault recovery scheme not being totally operational or not being operational at all, but does not result in any kind of disruption of the switch’s basic operation. Because of this fact, in Section 6, links and buses used in the fault recovery communication network are not taken under consideration during fault tolerance analysis.

5.2. The operation of FRCU

In the present subsection the operation of the FRCU is presented using pseudocode. The operation of the fault recovery subsystem depends on a communication protocol consisted of three control messages. These messages are exchanged between the FRCU and the SE’s:

1. **Notif Link failure (SE_ID, Direction)** is a notification message sent by a SE to the FRCU when the SE discovers that one of its links has gone down.

2. **Announce_SE_failure (SE_ID)** is a broadcast announcement sent by the FRCU to all SE’s when the FRCU concludes, by considering the combination of received faulty links reports, that there is a faulty SE.

3. **Announce_Link_failure (SE_ID)** is a broadcast announcement sent by the FRCU to all SE’s when a fault on the link connecting a SE to its corresponding output module is reported by the SE.

The purpose of the last two control messages is that on receipt of such messages, all SE’s can drop any cell destined to the specified SE, as there is no reason forwarding them; such cells will never manage to reach their destination, as their destination is not operational or unreachable.

The FRCU has to be aware of the grid topology. For this purpose an adjacency matrix (Adj_Matrix) is stored in the FRCU. A list of received reports (Reports_List) is also maintained in the FRCU, sorted by SE ID. The operation of the FRCU is described by the pseudocode below.

**On receipt of msg Notif Link failure (SE_ID, Direction)**

If SE_ID ∈ Probable_faulty_SEs then

  Remove SE_ID from Probable_faulty_SEs

If Direction = Output_module then

  Send msg Announce_SE_failure (SE_ID)

Else

  Corresponding_SE = Adj_Matrix [SE_ID, Direction]

  Add new record [SE_ID, Direction] in Reports_List

  Add Corresponding_SE in Probable_faulty_SEs

If 3 reports in Reports_list from all neighbours of Corresponding_SE ∈ Probable_faulty_SEs then

  Send msg Announce_SE_failure (Corresponding_SE)

5.3. The operation of each SE

Each SE preserves a list of SE’s announced to be faulty or having faulty link connecting them to the corresponding output module (Faulty_SEs_set). The pseudocode below presents the basic operation of each SE, as far as fault recovery is concerned.

**On link going down**

Send msg Notif Link failure (my_SE_ID, Direction)

**On receipt of msg Announce_Link_failure (SE_ID)**

Add SE_ID to Faulty_SEs_set

**On receipt of msg Announce_SE_failure (SE_ID)**

Add SE_ID to Faulty_SEs_set

In order to overcome the problem of faulty links connecting operational SE’s, care must be taken so that cells misrouted on purpose to another SE do not return to the transmitting SE. For this purpose, an additional bit is used in each cell’s tag, from now on called ‘NR’ (standing for Normal Routing). Value ‘0’ stands for normal routing, while value ‘1’ stands for inverse routing, i.e. routing done by firstly considering the 2nd bit of each pair of bits.

The following routing algorithm should be employed:

**Procedure Routing**

If Dest_SE_ID[1,2] (≠ SE_ID[1,2]) then

Route on (1,2)
Else
If Dest_SE_ID[3,4] ≠ SE_ID[3,4] then
    Route on (3,4)
Else
    Send to output module

**Procedure Route on (a,b)**
If NR = 0 then
If Dest_SE_ID[a] ≠ SE_ID[a] then
    If Dest_SE_ID[a] = 0 then
        Send to North
    Else (Dest_addr[a] = 1)
        Send to South
Else (Dest_SE_ID[a] = SE_ID[a], but
Dest_SE_ID[b] ≠ SE_ID[b])
    If Dest_SE_ID[b] = 0 then
        Send to West
    Else
        Send to East
Else (NR = 1)
    Inverse Route on (a,b)

**Procedure Inverse Route on (a,b)**
If Dest_SE_ID[b] ≠ SE_ID[b] then
    If Dest_SE_ID[b] = 0 then
        Send to West
    Else
        Send to East
Else (Dest_SE_ID[b] = SE_ID[b], but
Dest_SE_ID[a] ≠ SE_ID[a])
    If Dest_SE_ID[a] = 0 then
        Send to North
    Else (Dest_addr[a] = 1)
        Send to South

**Procedure Inverse Routing**
If Dest_SE_ID[1,2] ≠ SE_ID[1,2] then
    Inverse Route on (1,2)
Else
If Dest_SE_ID[3,4] ≠ SE_ID[3,4] then
    Inverse Route on (3,4)
Else
    Send to output module

**Procedure Check**
If cell destined to SE ∈ Faulty SEs_set then
    Discard cell
Else
    If cell destined to operational link then
        Send cell to that link
    Else
        If cell destined to faulty North/South link then
            Set NR: = 0
            Reverse Routing
            Check

Fig. 15. The fault recovery communication network, connecting the FRCU to SE’s.
If cell destined to faulty West/East link then
If both North/South links are down then
If both West/East links are down then
  Discard cell
Else
  Set NR: = 0
  Send to operational West/East link
Else
  Set NR: = 1
  Send to operational North/South link
If cell destined to faulty output-module link then
  Discard cell

The possible ‘paths’ between procedures that a cell can follow in a SE are shown in Fig. 16.

It should be noted that the FRCU is a single point of failure. If both the FRCU and a SE fail, then the neighbour SE’s will consider that there are only faulty links (not faulty SE), given that they don’t receive a relevant announcement from the FRCU. That would lead to cells being transferred from one SE to another forever, and subsequently lead to buffers’ overflow. So we need a way that cells destined to faulty SE’s would be discarded even though the FRCU has failed. For this purpose we can use ‘hop-counters’. Each cell has a hop-counter in its tag, which is checked on entering each SE. If the hop-counter is greater than a threshold it is discarded, otherwise it is increased by one. The threshold should be slightly higher than the diameter of the grid, so that diversions due to faulty links would not lead to discarding cells destined to operational reachable SE’s.

6. Fault tolerance analysis and comparison

In order to estimate the fault tolerance characteristics of GASA switches, we adopt the notion of ‘survival probability function’, introduced by Itoh in Ref. [23], which is also in compliance with the notion of ‘all-terminal reliability’ [25].

We consider all failures as random independent incidents and we define:
- Link Survival probability \( S_{\text{link}}(k) \) as the probability that all SE’s of the switch can communicate with each other, although there are \( k \) faulty links.
- SE Survival probability \( S_{\text{SE}}(k) \) as the probability that all operational SE’s of the switch can communicate with each other, although there are \( k \) faulty SE’s.

In order to calculate the values of the functions, the following equations are used:

\[
S_{\text{link}}(k) = \frac{\text{#DoC}}{L},
\]

\[
S_{\text{SE}}(k) = \frac{\text{#DoC}}{\#\text{SE’s}},
\]

where \( L \) is the number of links and DoC is the number of cases that ‘disruption of communication’ occurred. The switch fabric is considered disrupted when there is at least one pair of input and output modules, connected to operational SE’s, that cannot communicate through the fabric, as there is no path connecting these SE’s. We define an ‘operational SE’ to be a SE operating as described in Section 5.3 and being connected to operational input and output modules over operational links.

In Eq. (20), we consider all combinations of \( k \) faulty links that lead to disruption of communication between SE’s, divided by the total number of combinations of \( k \) links. It should be noted that only links connecting SE’s are taken into account, while neither links connecting SE’s to input and output modules nor links of the fault recovery communication network are taken into account, as they can never lead to disruption of communication as defined previously. Similarly in Eq. (21), we consider all combinations of \( k \) faulty SE’s that lead to disruption of communication between the rest (operational) SE’s, divided by the total number of combinations of \( k \) SE’s.

In Figs. 17 and 18 results from the SE survival probability function are presented for both 16×16 and 32×32 GASA switches.

In Figs. 19 and 20 results from the link survival probability function are presented for both 16×16 and 32×32 GASA switches. In the x-axis both the portion (percentage) (Figs. 17 and 19) and the absolute number (Figs. 18 and 20) of faulty links or SE’s is shown, in order to make comparisons easier.

We define Link Fail Probability as the probability that the
fabric is disrupted when there are k faulty links in the grid:

$$P_{\text{link}}(k) = 1 - S_{\text{link}}(k).$$  \hspace{1cm} (22)$$

Let $p(i)$ be the probability that the ith faulty link is the one that causes the disruption of the fabric. So we have:

$$P_{\text{link}}(k) = \sum_{i=1}^{k} p(i).$$  \hspace{1cm} (23)$$

From Eqs. (22) and (23), $p(i)$ can be calculated recursively. Now using Eq. (24):

$$a = \sum_{i=1}^{L} ip(i).$$  \hspace{1cm} (24)$$

we can calculate the average (or expected) number of faulty links that cause the fabric to disrupt. We consider the ‘a’ factor as the most important factor, as it indicates the value of the fault-tolerance scheme.

In Ref. [23] three Banyan-based fault-tolerant switch fabric architectures are compared:

1. the multi-plane network (Fig. 21), which in fact consists of 2 parallel Banyan networks,
2. the Benes network (Fig. 22), and
3. a MIN with redundant SE’s proposed by Itoh (Fig. 23).

In order to compare these 3 architectures to GASA the a factor is used, as well as a ‘cost efficiency factor’, $a/L$. All four architectures do not have the same size (number of links and number of SE’s), so they do not have the same hardware complexity. Consequently, the probability of having the same number of faulty links is different. So the deployment of a normalised factor, such as $a/L$, for comparison reasons is mandatory.

The results for $16 \times 16$ and $32 \times 32$ switches are shown in Figs. 24 and 25 for the a factor and the cost efficiency factor, respectively.

It can be noticed from Fig. 24 that GASA exhibits fault tolerant characteristics better than 2-plane Banyan network and Benes network. The architecture proposed in Ref. [23]
has a greater ‘a’ factor. The average number of faulty links that lead to switching disruption is 40–46% greater than GASA’s. However, when the necessary number of links of each architecture is taken under consideration and the normalised factor $a/L$ is considered, it can be noticed from Fig. 25 that GASA’s characteristics are much better: the cost efficiency factor is 73–113% greater than the one of the architecture proposed in Ref. [23]. This is because no redundancy is deployed and the number of links in GASA is far smaller than the number of links in other architectures. The numbers of internal links of all four architectures are given by the following equations:

$$L_{\text{plane Banyan}} = 2(n + 1)N,$$

$$L_{\text{Benes}} = 2(n - 1)N,$$

$$L_{[23]} = 3nN - 5N + 4,$$

$$L_{\text{GASA(square grid)}} = 4N - 4\sqrt{N},$$

$$L_{\text{GASA(non-square grid)}} = 4N - 6\sqrt{\frac{N}{2}},$$

where $n = \log N$.

The advantage of GASA against the other 3 architectures is due to the flexible re-routing performed in case of failures, instead of using redundant links and/or SE’s. This comes for a higher computational cost for advanced flexible routing in comparison to the basic non-fault-tolerant architecture and the other architectures presented and analysed in Ref. [23].

7. Conclusions

As line speeds increase ‘legacy’ architectures’ throughput, such as shared-bus and shared-memory, becomes inadequate. It is our belief that the industry will soon start deploying new architectures, such as space-division.

We have presented and studied a new grid-based ATM switch fabric architecture with minimal number of SE’s and
fault-tolerance capabilities. An analytical model, using no assumptions about the traffic pattern, was developed. The behaviour of the architecture was evaluated using extensive simulation. The results indicate that a speed-up factor of 2 in internal links is adequate for GASA to exhibit good performance without employing very large buffers, even under heavy loads. Even in the case where relatively large buffers were employed, there was not a major increase on cell delay, while the cell loss probability was decreased dramatically.

As far as fault-tolerance is concerned, the necessary additions to the basic architecture, the enhanced operation of the SE’s and the operation of the FRCU were presented. A simple protocol used in the communication between the FRCU and the SE’s was also defined.

In order to evaluate the fault tolerance, the notion of survival probability was used. Results were calculated and presented which demonstrate in a graphical way the behaviour of the GASA switch under concurrent failures. From these results, it can be noticed that the GASA switch has a remarkable behaviour regarding fault tolerance: even when 20% of the links have failed, the probability that all SE’s can communicate with each other is approximately 0.5. Even when 25% of the SE’s have failed, the probability that the operational SE’s can communicate with each other is over 0.65 in 16 x 16 switch and over 0.45 in 32 x 32 switch. GASA has also been compared to other fault tolerant switch architectures and it has been proven to be significantly better.

GASA is easily extensible to 3-dimensional grids. We strongly believe that such a 3-dimenisonal grid would exhibit even better characteristics, regarding both switching performance and fault tolerance, as it would form a more ‘strongly connected’ graph. A future work will include the evaluation of the 3-D architecture.

References


