Efficient ABR service engine for ATM networks

Y.Choi, S.Kang and S.Chong

Abstract: In the recent ATM Forum activities, considerable efforts have been focused on the available bit rate (ABR) service, which enables maximal link utilisation in the ATM network. An ABR service engine is presented, which provides optimal hardware solution for all functions of the ABR service algorithm. To compute congestion control information, the ABR service engine consists of an ER engine, a queueing connection (QC) estimation unit, and a cell decoder/encoder (CDE). To implement the algorithm efficiently, the new engine uses the following schemes. The ER is periodically computed in order to provide sufficient computation time. Through a periodical computation, a low cell delay is achieved when cells pass through the ABR service engine. Using a QC corrector, the implementation of the QC estimation unit is simplified. A register control block efficiently controls internal variables, and arithmetic units are designed for precise computation with simple architecture. Therefore, the ABR service engine is very small in size and provides high speed. In addition, it realises the computation of the congestion control information without a cell delay.

1 Introduction

Since most network applications cannot predict their own bandwidth requirements, they usually require a service that dynamically shares the available bandwidth among all active users. Such a service is called the ABR service [1, 2]. The important characteristics of an efficient congestion control algorithm for the ABR service include fast reaction to momentary congestion due to burst traffic, maximal link utilisation, fairness, and the low hardware complexity requirement. Various flow control schemes have been proposed for the ABR service, and they can be classified into two main classes: end-to-end rate-based schemes and link-by-link credit-based schemes. At the end of 1994, the ATM forum selected the rate-based control [3–7] as the flow control scheme for the ABR service for its simplicity. The congestion control lies at the heart of the general problem of traffic management of high-speed switching networks such as the ATM.

In this paper, an ABR service engine based on the simple, scalable, and stable explicit rate (ER) allocation algorithm [3], which prevents an excessive queueing delay or a deadlock when the incoming traffic to a specific link is heavier than the outgoing link capacity, is developed. The rate-based control information of the ABR service engine for congestion control is an explicit forward congestion indication (EFCI) marking, relative-rate markings, and an explicit rate (ER) marking. The relative-rate markings are no increase (NI) and congestion indication (CI).

Many architectures that have been proposed are slow in performance speed and have complex I/O interfaces due to the access of external memory. In order to arrive at a high-performance architecture, the algorithm reduces the complexity of computation, eliminates the necessity of per-VC accounting and does not access an external memory. Also, a periodical ER computation obviates a redundant arithmetic unit and diminishes the occurrences of a cell delay. Most ideas for the design originate from this scheme. If the ER computation starts whenever the resource management (RM) cell arrives at the cell decoder, a buffer will store cells during the ER computation. A disadvantage of this scheme is that it makes the architecture complex and causes a cell delay. Since the periodic scheme provides sufficient time for an ER computation, these problems can be solved. That is, a small arithmetic architecture in size can be adopted. Since the ER is stored after computing, the ABR service engine only writes beforehand the prepared ER in the RM cell, without the cell delay whenever an RM cell arrives at the cell decoder. It can achieve simpler and faster architecture without affecting performance.

2 Algorithm of ABR service engine

It is very difficult to convert an algorithm completely into a circuit, since a circuit has many limited conditions, such as computation delay, data type and timing, etc. So, the simple, scalable and stable ER allocation algorithm is modified in order to implement the ABR service engine into the hardware.

Each RM cell contains a rate at which the sender would currently like to transmit data. Such a value is called the explicit rate (ER). As the RM cell passes through the receiver, the congested switches may reduce the ER. That is, the ER value is used to limit the allowed cell rate (ACR) of a source to a specific value. The ER computation algorithm is shown in Fig. 1. Since the ER is computed by the periodical scheme, the QC correction process is added.

QC represents the number of the connection that can cause queueing in each switch. The basic interval of all parts for a QC prediction is W. A detailed prediction originates from QC_1, the source number of QC. When a new

IEE Proceedings online no. 20010622

DOI: 10.1049/ip-cds:20010622

Paper first received 27th March 2000 and in revised form 9th July 2001

Y. Choi is with the Display Division, LG Electronics Inc., Seoul, Korea

S. Kang is with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea

S. Chong is with the Department of Electrical Engineering and Computer Science, KAIST, Daejon, Korea

connection occurs, the connection is assumed to relate with queueing [3]. The QC prediction and correction algorithm is shown in Fig. 2.

```
if (every T period){
    QL_ave = sum of queue lengths / queue count
// g_TH is the criterion in selection of A and B
    if (g_TH < queue length){
        if (system start){
// A and B are coefficients to react quickly when starting the ATM
            A = A0, B = B0
        }
        else {
            A = A1, B = B1
        }
    }
    QC_temp = QC_estimation + QC_corrector
// q_T is the target of the queue length which must be maintained
// ER_pre is the ER value computed by the ER engine at a previous time
// Δ is computational interval
    ER = ER_pre - A/QC_temp (QL_ave - QL_ave_pre) - BΔ/QC_temp (QL_ave - q_T)
    ER is limited between link-speed and 0
}
```

Fig. 1 *ER computation algorithm*

```
if (receive the forward RM cell sent by source){
// δ is an averaging factor between 0 and 1
    if(RM(CCR) - RM(MCR) > ER engine (δ × ER)){
// N_RM is the number of cells per forward RM cell
        QC_1 = QC_1_previous + N_RM / (W × RM(CCR))
    }
}
if (every W period){
    QC_temp = QC_previous + QC_corrector
    QC = QC_temp × λ + QC_1 × (1 - λ), 0 < λ < 1
    QC is limited between total TC and 0
    QC_corrector = QC_1 =
}
if (increase TC){
    QC_corrector = QC_corrector + the increase of TC
}
```

Fig. 2 *QC prediction and correction algorithm*

The switch may set the EFCI in an ATM data cell header as it passes forward. This will cause the destination end system to set the congestion indication (CI) bit in a backward RM cell. In addition, the switch may directly set the CI or no increase (NI) bit of a passing RM cell. If the bit is set in a forward RM cell, then that bit will remain set in the corresponding backward RM cell when a turnaround occurs at the destination. To achieve a result most rapidly, a switch may generate a backward RM cell with the CI or the NI set, rather than wait for a passing backward RM cell. If an input cell is a backward RM cell sent by a source and it satisfies the CRC check, the ABR service engine prepares to write the congestion information in the RM cell. When the subtraction of an ER and an MCR obtained from the RM cell is bigger than the ER from the ER engine, the ER from the ER engine is written in the RM cell. After writing all the congestion information, the ABR service engine generates a new CRC value for the modified RM cell.

3 Architecture of ABR service engine

The architecture of the ABR service engine shown in Fig. 3 is divided into several parts: the cell decoder/encoder (CDE) unit, the QC estimation unit, the ER engine and other control logics. The QC estimation unit periodically computes the QC for the ER engine, and the ER engine periodically computes a new ER, which will be written in the RM cell. The CDE unit includes a cell decoder and a cell encoder. A cell decoder detects an RM cell during a cell flow and reads the information for each module, but a cell encoder writes new congestion control information in the cell. The EFCI is marked in the data cell, but the relative rate and the ER are written in the RM cell. To detect an error in the ATM traffic, the cell decoder has a CRC checker, while the cell encoder has a CRC generator for the modified RM cell. The universal test and operations physical layer interface for ATM (UTOPIA) [8], that defines the interface between the physical layer and the upper layer modules such as the ATM layer, is obeyed in the I/O of the CDE unit.

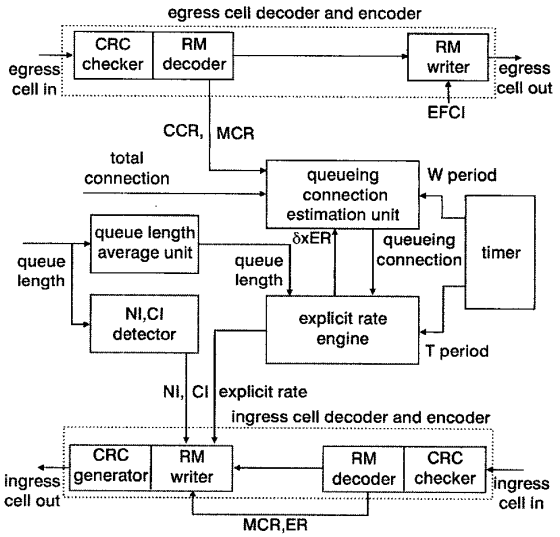

Fig. 3 *ABR service engine*

As the main number system of the ABR service engine, the 32-bit floating-point IEEE 754 single precision [9] is adopted to reduce errors in complex computations. Two's complement and rate-format number systems are adopted in order to achieve a simple architecture. When necessary, these formats are converted into floating-point formats for internal computation. The rate-format is represented as $[2^e(1 + m/512)] \times nz$ (cells/s), with a 5-bit exponent (e), a 9-bit mantissa (m) and a 1-bit nonzero flag (nz).

The I/O bus interface controller can read and write the parameter values in the register file. New parameter values that are obtained from a simulation are written in the register file for normal operations in other environments, and the register file is read for monitoring internal operations of the system. The queue length obtained from the external queue is used for congestion information and the ER engine. The relative rate and the EFCI marking result from comparing current queue lengths with a threshold.

Fig. 4 shows the architecture of the ER engine. The process of computing the ER in the ER engine starts every T period. After the ER computation, the new ER is updated periodically. The ER engine requests a number of floating-point arithmetic units, because the ER formula has

four multiplications, three divisions and five additions. Implementing only a multiplier, a divider and an adder, and reusing them make a simple architecture. Although this makes the computation slower, the ER computation time is sufficient due to a periodic scheme. Since each arithmetic unit operates independently in the ER engine, the parallel computation is possible. Therefore, the proper order of computation utilising all units at the same time can reduce the computing time. Finally, the ER is prepared for the RM cell after being converted into the rate-format. Also, the ER engine computes the value that the QC estimation requires.

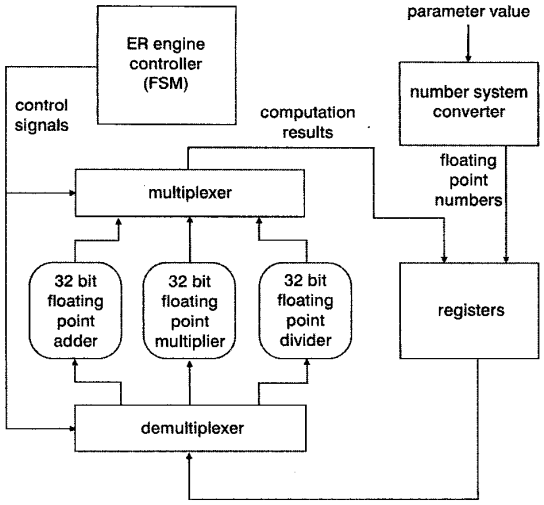

Fig.4 *Re-using scheme of ER engine*

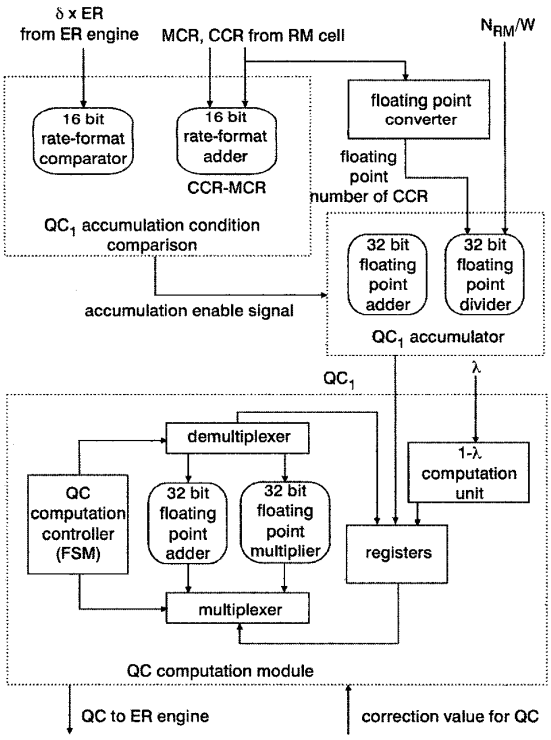

Fig.5 *QC estimation unit*

The QC estimation unit shown in Fig. 5 acts as a low pass filter, which smoothes the extreme variation of QC. The QC estimation assumes queueing in the output port. Since the QC estimation unit treats all connections of an ABR class at once, this reduces the complexity of control and the waste of storage. The QC estimation unit is divided into two parts: a QC_1 accumulation and a QC computation part. While QC is computed, the QC estimation cannot sense a variation of total connection (TC). The TC variation obtained from the QC corrector during W period is used in order to compute the next QC precisely. Also, the QC correction is applied to QC used in the ER engine. The last step of the QC estimation is to limit the QC, because it must be smaller than the TC. The arithmetic unit is designed for reducing the area overhead while satisfying the timing constraints and precision requirements. Since the floating-point division and addition cannot be completed in a cell time, it becomes a problem when the RM cell continuously and endlessly arrives. However, this problem is solved by inserting registers between the divider and the adder in the pipeline scheme.

The cell decoder/encoder (CDE) unit directly treats cells as the intermediate between a network and a system outside of the architecture. It is classified into two parts, egress and ingress. The operations of the egress CDE shown in Fig. 6 is as follows. The cell decoder of the egress CDE receives the latest information from the RM cell and checks the CRC to decide whether the information is correct or not. The cell decoder recognises whether the input is a data cell or an RM cell. The transfer of a cell is synchronised by the start of cell (SOC) signal of UTOPIA. This signal is asserted when the data transfer path contains the first byte of a cell. If the input is a data cell and if the network is congested, the cell encoder will mark EFCI in the cell header. If the input is an RM cell, the preparation for the QC_1 computation is launched. The QC_1 accumulator uses only the data of the RM cell that is at the forward direction and sent by the source. The others are ignored in the cell decoder. After completing all identifications, the cell decoder sends an RM identification signal to the QC_1 accumulation part.

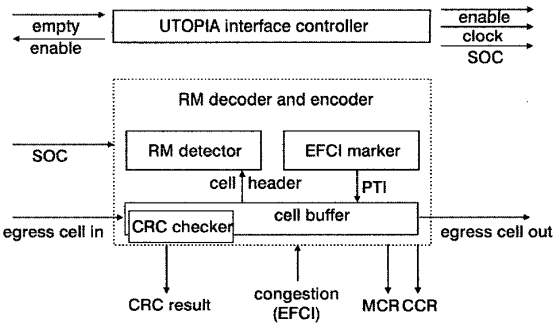

Fig.6 *Egress CDE*

The ingress CDE shown in Fig. 7 is similar to the egress CDE, but in the ingress, the cell encoder is more complex than the cell decoder, because the cell encoder writes three kinds of information about the NI, the CI, and the ER, and generates a new CRC. The identification of the RM cell is the same as that of the egress cell decoder except for the direction. The ingress cell decoder deals with not the forward but the backward RM cell and reads ER instead of CCR. This information is sent to the cell encoder. After the rate-format adder adds the CCR of the RM cell and the ER of the ER engine, if the sum is less than the ER of the RM cell, the cell encoder will write the sum in the RM cell. If the current queue length is greater than the threshold of the congestion, the relative-rate will be set to warn

the occurrence of the congestion to the source. The cell encoder completes the marking for a rate control and generates a new CRC.

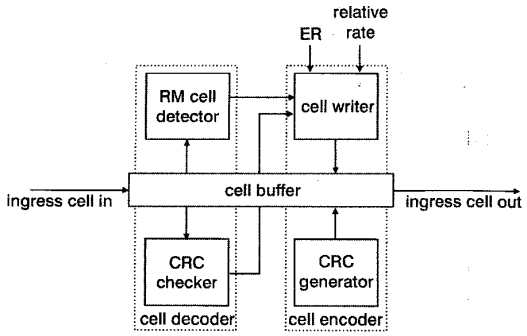

Fig. 7 *Ingress CDE*

4 Implementation issues

In order to implement a high-performance architecture the following points are considered. The main focus is to obtain high-speed operations. The periodical computation scheme can solve this problem, but it raises a new problem that the QC estimation unit cannot sense the variation of TC during a periodical computing. Therefore the ABR service engine introduces an additional block, the QC corrector, to maintain the exact QC. Moreover, since many data types are used, the number system convert is used to transfer data among modules easily.

The original functions of the QC estimation unit are to compute QC from QC_1 and increase the QC as much as the amount of the TC variation simultaneously. Since it is very difficult to implement the two functions, they are divided into a QC estimation unit and a QC corrector. The QC corrector senses the variation of TC separately during the W period. Fig. 8 illustrates the architecture of a QC corrector. The sum of the QCs from the QC estimation unit and the temporary QC from the QC corrector is used in the QC estimation unit and the ER engine. The module needs an additional adder for correcting the QC but the adder is not necessary due to a re-use scheme.

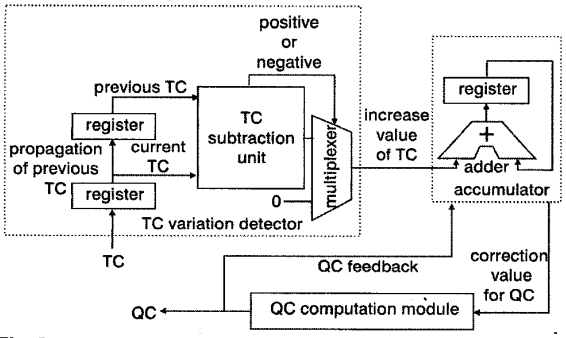

Fig. 8 *QC corrector*

A floating-point is the main number system in internal modules for precision, but other number systems are also used due to the simplicity of the architecture and fast operations in some modules. Therefore, there are three number system conversions, i.e. converting two's complement to floating-point, converting floating-point to rate-format and converting rate-format to floating-point.

Since the values of the register can be changed according to the environment by the I/O bus controller, the ABR service engine is compatible with other link-speeds, although this feature increases the number of registers. Besides, the register file provides the function of observing inner operations. The observation becomes possible by reading the registers, since an internal variable is stored at the registers. When the network provider sets a system, the register file is initialised through the I/O bus controller.

There are two arithmetic units in this architecture, one for a floating-point and the other for a rate-format. The floating-point unit is composed of an adder, a multiplier and a divider. The rate-format unit is an adder. Although its size increases, the multiplier adopts the Booth algorithm [10–12] for fast multiplication. The divider uses the non-restoring algorithm [11, 12] because it is the most efficient in size and speed. The sequential architecture can use only one adder instead of an adder array. This requires much fewer adder cells and does not have as many wasteful activities in the sign bits.

5 Results

The architecture of the ABR service engine is synthesised to a gate-level using Synopsys. The schematic diagram of the synthesis result using the 0.5μm Samsung library is shown in Fig. 9. Table 1 shows the gate level synthesis results of the ABR service engine. The total number of cells was 35693, and the critical path delay was 20.51 ns. We use the EPF10K50RC240 device of Atera FLEX10k series [Note 1] for an FPGA implementation, since the number of the cells and the I/O pins of the ABR service engine are 35693 and 189, respectively. In order to compile, the MAX+PLUS II CAD [Note 1] tool is used and an FPGA board is made for verification. By providing an interface between the FPGA board and a PC, the virtual network condition is programmed on a PC, and it verifies the FPGA board.

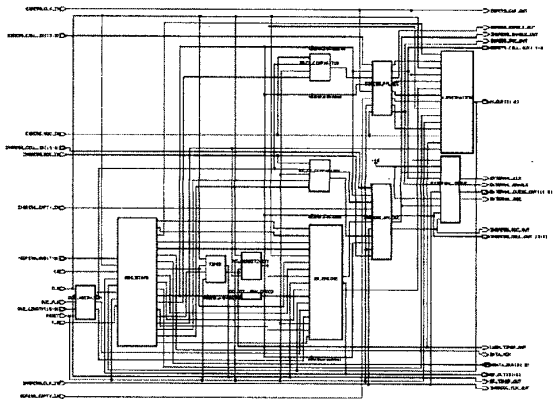

Fig. 9 *Synthesis result of ABR service engine*

Table 1: Area and delay of ABR service engine

	Number of cells	Critical delay
ER engine	9955	17.28 ns
QC estimation unit	9831	20.30 ns
CDE	7225	17.60 ns
Register file	5170	1.41 ns
QC corrector	1108	20.51 ns
All other logic	2404	–
Total	35693	20.51 ns

Note 1: Altera Co.: 'FLEX 10K device family' available from http://www.altera.com/html/products/f10k.html, 1999

The verification process is as follows. The input and output data of the verification are obtained by the program based on [3]. The inputs of the ABR service engine are an egress cell, an ingress cell, a queue length and a TC. The number of all input patterns is approximately 10 million. When the ABR service engine is initialised, the queue length is also initialised. To meet a verification condition, the queue length is increased, and the queue length affects the ER and the QC. Since the ABR service engine changes the ER value so that the queue length can be equal to q_T, the queue can reach the steady state by the new ER value. The ER computation also reaches the steady state by the queue of the steady state. The TC is increased by a new connection. The increase of TC is handled in the QC corrector, and the corrected QC is used for the ER value to maintain the steady state. The queue and the ER value reach the steady state again. Fig. 10 shows the process of the steady state of the ER value using the program based on [3] and the FPGA, and Fig. 11 shows the process of the steady state of the QC. In spite of the variation of inputs, the ABR service engine always reaches the steady state in the simulation. Also, the simulation results from using the program based on [3] and from a hardware implementation are identical.

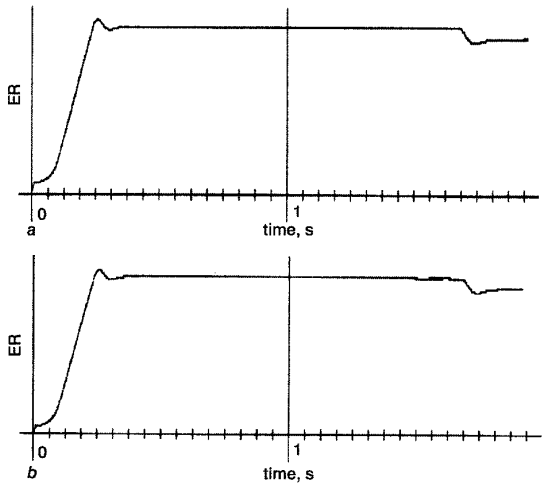

Fig. 10 *Process of steady state of ER value*
a Simulation result using program based on [3]
b Simulation result using FPGA

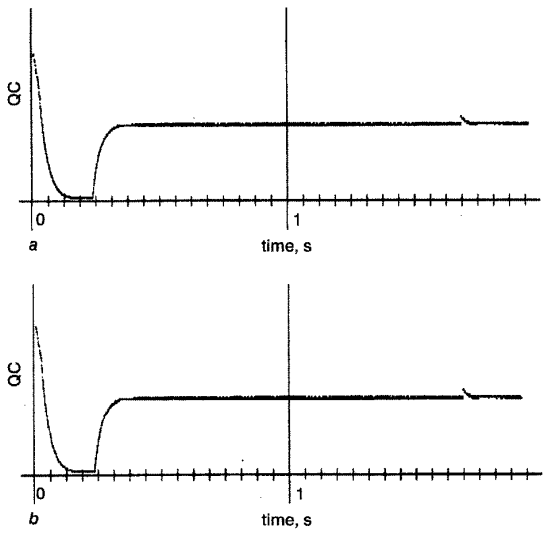

Fig. 11 *Process of steady state of QC*
a Simulation result using program based on [3]
b Simulation result using FPGA

6 Conclusions

In this paper, an efficient architecture of an ABR service engine, which is based on a simple, scalable and stable ER allocation algorithm, has been developed. This architecture implements the algorithm optimally without performance degradation using the following features. First, due to a periodic scheme, the architecture of the ABR service engine is simple and the cell delay is short. That is, since this scheme can provide sufficient computation time, the arithmetic units can be simple in spite of using the floating-point number system. Since the ABR service engine does not need to store the cells in the buffer for the ER computation, the cell delay cannot occur. Second, the number system of the arithmetic unit is the floating-point number for reducing error due to the complex computation. This number system can guarantee stable operations of the system. Third, a QC corrector and a number system converter solve the difficulty of the implementation. The ABR service engine was implemented and verified using an FPGA. The implementation showed that the new architecture is area efficient and achieves reasonable clock frequencies.

7 Acknowledgments

This work was supported by Samsung Electronics Inc.

8 References

1 SATHAYE, S.: 'ATM Forum traffic management specification'. Version 4.0, 1996
2 BONOMI, F., and FENDICK, K.W.: 'The rate-based flow control framework for the available bit rate ATM service', *IEEE Netw.*, 1995, **9**, (2), pp. 25–39
3 CHONG, S., LEE, S.-H., and KANG, S.: 'A simple, scalable and stable explicit rate allocation algorithm for max-min flow control with minimum guarantee', *IEEE/ACM Trans. Netw.*, 2001, **9**, pp. 322–335
4 WONG, M.K., and BONOMI, F.: 'A novel explicit rate congestion control algorithm'. Proceedings of IEEE GLOBECOM'98, 1998, Vol. 4, pp. 2432–2439
5 KALAMPOUKAS, L., VARMA, A., and RAMAKRISHNAN, K.K.: 'An efficient rate allocation algorithm for ATM networks providing max-min fairness'. Technical report UCSC-CRL-95-29, Computer Engineering Dept., University of California, Santa Cruz, 1995
6 CHARNY, A., RAMAKRISHNAN, K.K., and LAUCK, A.: 'Time scale analysis and scalability issue for explicit rate allocation in ATM networks', *IEEE/ACM Trans. Netw.*, 1996, **4**, pp. 569–581
7 JAIN, R., *et al.*: 'ERICA switch algorithm: A complete description'. ATM Forum/96-1172, 1996
8 ATM Forum technical committee: 'UTOPIA specification level 1'. Version 2.01, 1994
9 IEEE: 'IEEE standard for binary floating-point arithmetic'. IEEE 754-1985
10 BOOTH, D.: 'A signed binary multiplication technique', *Q. J. Mech. Appl. Math.*, 1951, **4**, (2)
11 KOREN, I.: 'Computer arithmetic algorithms' (Prentice-Hall, 1993)
12 HWANG, K.: 'Computer arithmetic: principles, architecture, and design' (Wiley, 1979)