AN EFFICIENT ABR SERVICE ENGINE FOR ATM NETWORK

Youngin Choi — Display Division, LG Electronics Inc., Kumi, Korea

Sungho Kang — Dept. of Electrical Engineering, Yonsei University, Seoul, Korea

Song Chong — Dept. of Electronic Engineering, KAIST, Daejon, Korea

ABSTRACT

In recent ATM forum activities, considerable efforts have been focused on an Available Bit Rate (ABR) service, which enables maximal link utilization in the ATM network. In this paper, we present an efficient architecture of an ABR service engine, which includes all functions required in the ABR service algorithm. The new architecture of the ABR service engine is very small in size and high speed by computing the congestion control information without cell delay.

I. 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 an 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 utilization, fairness, and low hardware complexity requirement.

Various flow control schemes have been proposed for the ABR service, and they can be classified into two classes, the end-to-end rate-based schemes and the link-by-link credit-based schemes. In late 1994, the ATM forum selected the rate-based control [3,4,5] as the flow control scheme for the ABR service due to 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.

The congestion arises when the incoming traffic to a specific link is heavier than the outgoing link capacity, in which case the buffer could overflow, causing an excessive queuing delay or even a deadlock in the network. For this problem, the simple, scalable and stable Explicit Rate (ER) allocation algorithm [6] is developed. In [6], 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 a No Increase (NI) and a Congestion Indication (CI).

Previous architectures are slow in performance speed and have complex I/O interface due to the access of external memory. In this paper, to overcome the problems of the hardware complexity and the speed, complex computation is reduced by removing per-VC accounting and external memory access. Also, a periodical ER computation is used to reduce the hardware and the occurrences of a cell delay since the periodic scheme provides sufficient time for an ER computation. the new ABR service engine only writes 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.

II. ABR SERVICE ALGORITHM

To implement an efficient architecture, a simple, scalable and stable ER allocation algorithm is devised. The pseudo code for the ER computation is shown in Figure 1. Each RM cell contains a rate at which the sender would currently like to transmit data. Such a value is called an Explicit Rate (ER). As the RM cell passes through to 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. A QL_{ave} is the average of the queue lengths during T seconds, which is a multiple of unit time that takes a cell to pass through the ABR service engine. A and B are coefficients for the ER formula and have two values in order to react quickly when starting the ATM, and g_{TH} is the criterion in selecting of A and B. 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. Since the ER is computed by the periodical scheme, the QC correction process is added. A QC represents the number of the connection that can cause queuing in each switch.

To predict the QC_l, the source number of QC, the number of forward RM cells is $W \times RM(CCR)/N_{RM}$ per connection at the input where NRM is the number of cells per forward RM cell and W is the interval. The QC can be predicted by continuous summations of inverse values of the average of the forward RM cells for all connections. The pseudo-code for QC prediction and QC correction is shown in Figure 2. Since QC may be oscillated according to W periods, QC prediction includes the feature of a low pass filter for smoothing the oscillation. A QC is corrected by identifying the Total Connection (TC). When a new connection occurs, the connection is assumed to relate with queuing [6]. λ is an averaging factor between 0 and 1.

```
if ( every T period ){
        $QL_{ave}$ = sum of queue lengths / queue count
        if ( $g_{TH}$ < queue length ){
                if ( system start ){
                        A = A0,B = B0
                }
                else {
                        A = A1,B = B1
                }
        }
        $QC_{temp} = QC_{estimation} + QC_{corrector}$
        $ER = ER_{pre} - \frac{A}{QC_{temp}}(QL_{ave} - QL_{avepre}) - \frac{B\Delta}{QC_{temp}}(QL_{ave} - q_T)$
        ER is limited between link-speed and 0
}
```

Figure 1. ER computation algorithm.

```
If ( every W period ) {
        $QC_{temp} = QC_{previous} + QC_{corrector}$
        $QC = QC_{temp} \times \lambda + QC_l \times (1 - \lambda), \quad 0 < \lambda < 1$
        QC is limited between total TC and 0
        $QC_{corrector} = QC_l = 0$
}
if ( increase TC ) {
        $QC_{corrector} = QC_{corrector}$ + the increase of TC
}
```

Figure 2. QC prediction and QC correction algorithm

The switch may set the EFCI in an data cell header as it passes forward. This causes 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 it will remain 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 rather than waiting 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. After writing the information, the ABR service engine generates a new CRC value for the modified RM cell.

III. ABR SERVICE ENGINE ARCHITECTURE

The architecture of the ABR service engine is shown in Figure 3. 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. A timer periodically enables them to compute. A cell decoder detects an RM cell during a cell flow and reads the information for each module, but the 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. On the other hand, the cell encoder has a CRC generator for the modified RM cell. The Universal Test & Operations Physical Layer Interface for ATM (UTOPIA) [7] 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 cell decoder and encoder.

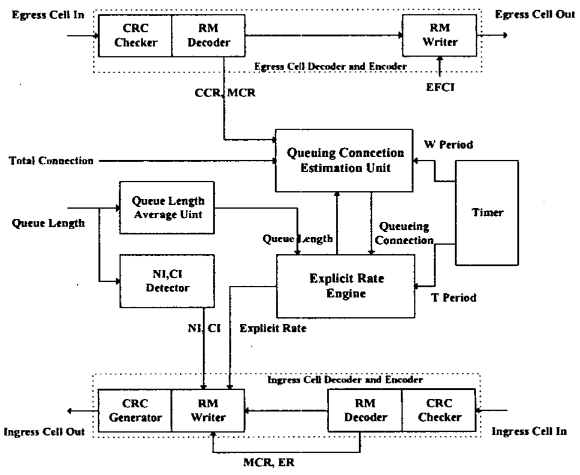


Figure 3. Architecture of ABR service engine.

As the main number system of the ABR service engine, the 32-bit floating-point, IEEE 754 single precision, is adopted to reduce errors in complex computations. A two's complement and a rate-format number systems are adopted to achieve at a simple architecture. So other formats must be converted into the floating-point formats for internal computation. The rate-format [1] is represented in a binary floating-point representation employing a 5-bit exponent (e), a 9-bit mantissa (m) and a 1-bit nonzero flag (nz). That is, rate-format is $[2^e(1+m/512)] \times nz$ (cells/seconds).

The I/O bus interface controller can read and write the parameter values in the register file. Therefore, the proper order of computation utilizing all units at the same time can reduce the computation time. The time needed to obtain the ER is nearly the same as the time needed to pass three dividers. This process is repeated until the ER computation is completed.

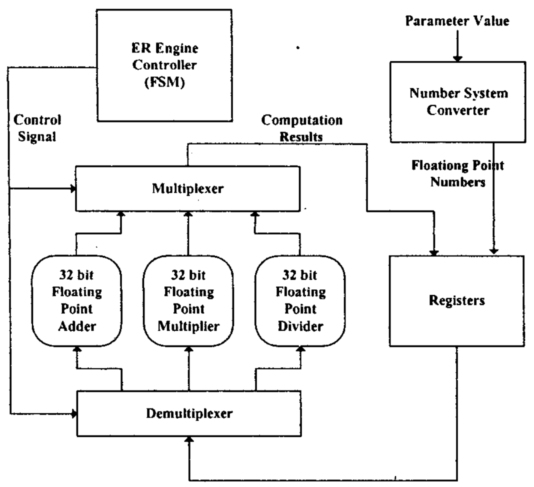


Figure 4. ER engine.

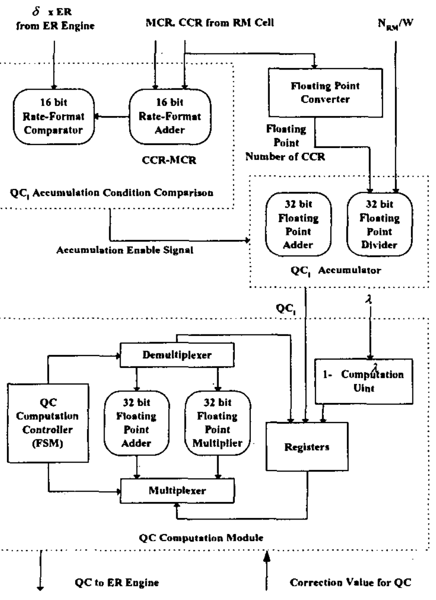
Figure 4 shows the architecture of the ER engine. The process of computing the ER in the ER engine starts every T period. Since T is a comparatively long time in arithmetic, ER computation can be sufficiently completed during the T period. After the ER computation, the new ER is updated at a register periodically. The ER engine requests a number of floating-point arithmetic units, because the ER formula has four multiplications, three divisions, and five additions. Modules requiring a number of arithmetic operations have only one unit for each arithmetic operation and reuse it. This method makes the control logic more complex and the computation slower. But the architecture using all arithmetic units like the ER formula is more complex than one using a unit for each arithmetic operation, and the ER computation time of the reuse scheme is sufficient due to a periodic and parallel scheme. Since each arithmetic unit operates independently in the ER engine, the parallel computation is possible. Therefore, the proper order of computation utilizing all units at the same time can reduce the computation time. The time needed to obtain the ER is nearly the same as the time needed to pass three dividers. MUX and DEMUX are at each input and output of the arithmetic units in order to implement the reuse scheme. This process is repeated by the controller until the ER computation is completed.

The ER cannot be higher than the link-speed. Finally, the ER is prepared to be written in the RM cell after being converted into the rate-format. Also, the ER engine computes the value that QC estimation requires. This value is obtained by multiplying ER by a comparison margin, δ. The ER engine requires a few parameters for the computation. The constant values obtained by the simulation are A, B, T and q_T. External input values are the sum of the queue lengths, the queue number for the average of the queue lengths and the QC delivered from the QC computation unit. The internal feedback variables are the ER and the average of the queue lengths.

The QC estimation assumes the queuing in the output port. Also, the QC estimation unit does not treat the information of each connection individually but treat all connections of an ABR class at once. The former complicates the control and wastes storage elements, but the latter can prevent the problems. QC estimation unit is divided into two parts, namely, a QC_I accumulation and a QC computation part. The QC estimation unit operates every W period.

The QC_I computation process begins whenever the RM cell arrives at the ABR service engine, and QC_I is reset every W period. The QC_I accumulation part is again divided into a QC_I accumulation condition comparison and a QC_I accumulator. The QC_I accumulation condition comparison part compares $\delta \times ER$ with the result taken from the subtraction of the Minimum Cell Rate (MCR) from the Current Cell Rate (CCR). CCR and MCR are fetched from the forward RM cell, and $\delta \times ER$ is sent from the ER engine, i.e, CCR-MCR > $\delta \times ER$. To simplify the architecture, an adder and a comparator are the arithmetic units for a 16-bit rate-format that is simpler than a 32-bit floating-point. Also, since $\delta \times ER$ is a floating-point, it must be converted into the rate-format. Simultaneously, the QC_I accumulator divides N_{RM}/W, which is a constant computed by the simulation, by the CCR that is converted into the floating-point. The QC_I accumulator and the QC computation unit use the floating-point number for precision due to the feedback of QC_I and QC. Since the floating-point division is slower than other arithmetic units in speed, QC_I has to be previously computed in order to be used in the QC_I accumulator when the RM cell arrives at the ABR service engine. After a division, the results of the QC_I accumulation condition comparison and the CRC check for detecting errors in the RM cell are prepared to accumulate QC_I, and they choose whether to use or discard the division result in the QC_I addition. The floating-point division and addition can not be completed in a cell time. It becomes a problem when the RM cell continuously arrives. But since the division and the addition can respectively be completed in a cell time, the used pipeline solves the problem. That is, registers are inserted between the divider and the adder in the pipeline scheme.

The block diagram of the QC estimation unit is shown in Figure 5. The QC computation part acts the role of a low pass filter feature which smoothes the extreme variation of QC. The new QC is the sum of λ ratio of the previous QC and 1-λ ratio of QC_l that is measured in the current section. Because λ is a floating-point number between 0 and 1, the exponent of λ is a constant. So, 1-λ can be computed by a specific block that has an inverter and a shifter. That is, 1-λ is obtained by inverting and normalizing the fraction part of λ. Though this scheme may cause trivial errors, it guarantees sufficient precision for the QC computation. Besides, the block that has the inverter and the shifter is faster and simpler than the floating-point adder. The QC computation architecture is similar to the ER engine, but it lacks a divider. While QC is computed, the QC computation cannot sense a variation of Total Connection (TC). That is, as TC increases, QC must increase as much as the degree to which the variation of TC increases. 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 computation is to limit the QC, because it must be smaller than the TC. After completing the computation, the QC is delivered to the ER engine



δ x ER from ER Engine
MCR, CCR from RM Cell
N_RM/W
16 bit Rate-Format Comparator
16 bit Rate-Format Adder
CCR-MCR
QC_l Accumulation Condition Comparison
Floating Point Converter
Floating Point Number of CCR
32 bit Floating Point Adder
32 bit Floating Point Divider
Accumulation Enable Signal
QC_l Accumulator
QC_l
λ
Demultiplexer
1-λ Computation Unit
QC Computation Controller (FSM)
32 bit Floating Point Adder
32 bit Floating Point Multiplier
Registers
Multiplexer
QC Computation Module
QC to ER Engine
Correction Value for QC

Figure 5. QC estimation unit

The cell decoder and the cell encoder directly treat cells as the intermediate between a network and a system outside of the architecture. It is divided into two parts, egress and ingress. Both parts are similar in function but different in usage. That is, they read and write different contents. The cell decoder and the cell encoder obey the UTOPIA interface to transmit data among the network and the CRC-10 to prevent error in an RM cell.

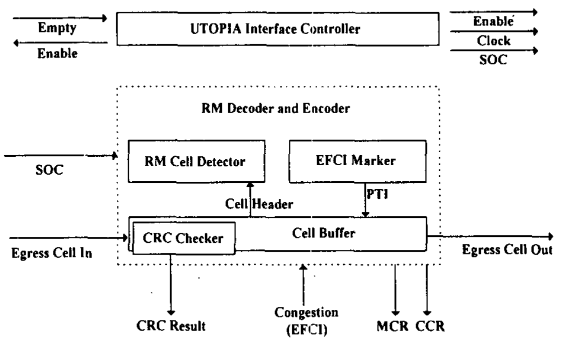


Figure 6. Egress cell decoder and encoder

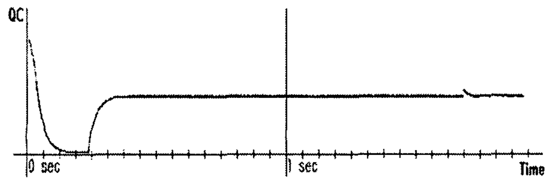
The cell decoder receives the latest information from the RM cell and checks the CRC to decide whether the information is correct or not. This information is sent to the QC estimation unit. The cell decoder mainly operates along with the QCl accumulator. The specific functions of the cell decoder and the encoder are as follows. The cell decoder recognizes whether the input is a data cell or an RM cell. The transfer of a cell is synchronized 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. The Payload Type Indication (PTI) in a cell header has information on cell types and EFCI congestion. After the SOC is asserted, the cell decoder reads the PTI in the cell header. When the PTI is "110", the input is an RM cell. Otheriwise it is a data cell. The architectures of the egress cell decoder and the encoder are shown in Figure 6. In the case of the data cell, if the network is congested, the cell encoder will mark EFCI in the PTI of the cell header. The EFCI value is obtained from comparing the current queue length with g_{EFCI}, which is the congestion threshold that has been used for rate control in the past. If the input is an RM cell, the preparation for QC_l computation is launched. The QCl 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. The cell decoder can recognize the RM cell from a Direction (DIR) and a BECN cell (BN) of a message type. After completing all identifications, the cell decoder sends an RM identification signal to the QC_l accumulation part. The QC_l computation starts by asserting this signal. After the CCR and the MCR are obtained from the RM cell one by one, these are sent to the QC_l accumulator. But if an error occurs during the CRC check, all operations will be ignored.

The ingress cell decoder and encoder is similar to those of the egress. But in the ingress, the cell encoder is more complex than the decoder, because the cell encoder writes three kinds of information about the NI, the CI and the

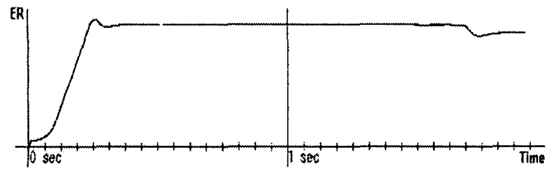
ER and generates a new CRC. The identification of the RM cell is the same as that of the egress cell decoder except for the direction. The ingress cell decoder deals with not forward but the backward RM cell and reads ER instead of CCR. This information is sent to the cell encoder. After the rate-format adder adds the CCR of the RM cell and the ER of the ER engine, the sum is compared with the ER of the RM cell. That is, RM cell (ER) > ER engine (ER) + RM cell (MCR). If this is true, 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 a congestion to the source. The cell encoder completes the marking for a rate control and generates a new CRC. However, if an error occurs during the CRC check, all operations will be cancelled.

IV. RESULTS

The ABR service engine is implemented using 0.5 (μm) Samsung library. The number of gates are 35,693 and the number of I/O pins are 189, respectively. The critical path of the engine is 20.51ns.



(a) Steady state of the ER



(b) Steady state of the QC

Figure 7. Verification of the ABR Service Engine

In order to verify the ABR service engine, the input and output data of the verification are obtained by the program based on [6]. The inputs of the ABR service engine are an egress cell, an ingress cell, a queue length and a TC. The number of simulation input patterns is about 10,000,000. When the queue length is increased, the queue length affects the ER and the QC. Since the ABR service engine changes the ER value, the queue can reach the steady state by the new ER value. The ER computation also reaches 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. In spite of the variation of inputs, the ABR service engine always reaches the steady state as shown in Figure 7.

V. CONCLUSION

A new efficient architecture of the simple, scalable and stable ER allocation algorithm for the ABR service is proposed. The features of this architecture are as follows. First, because of the usage of a periodic scheme, the architecture of the ABR service engine is simple and the cell delay is short. Second, the number system of the arithmetic unit is the floating-point number for reducing error due to the complicate computation. This number system can guarantee the stable operation of the system. Third, a QC corrector and a number system converter solve the difficulty of the implementation. Consequently, the ABR service engine can achieve a small-sized, simple architecture with a low cell delay.

Acknowledgement

This work was supported by Samsung Electronics Inc.

References

[a] S. Sathaye, ATM forum Traffic Management Specification, Version 4.0, Feb. 1996.
[b] F. Bonomi and K. W. Fendick, "The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service," IEEE Network, vol. 9, no. 2, pp. 25-39, 1995.
[c] M. K. Wong and F. Bonomi, "A Novel Explicit Rate Congestion Control Algorithm," Proc. of GLOBECOM, vol. 4, pp. 2432-2439, 1998.
[d] A. Charny, K. K. Ramakrishman and A. Lauck, "Time Scale Analysis and Scalability Issue for Explicit Rate Allocation in ATM Networks," IEEE Trans. on Networking, vol. 4, pp. 569-581, 1996.
[e] Song Chong, Sang-Ho Lee and Sungho Kang, "A Simple, Scalable and Stable Explicit Rate Allocation Algorithm for MAX-MIN Flow Control with Minimum Guarantee," IEEE Trans. on Networking, to be published.
[f] "UTOPIA specification Level 1," The ATM forum Technical Committee, Version 2.01, March 21, 1994.