Buffer Management for Shared-Memory ATM Switches

Mutlu Arpaci and John A. Copeland, Georgia Institute of Technology

Abstract

In the shared-memory switch architecture, output links share a single large memory, in which logical FIFO queues are assigned to each link. Although memory sharing can provide a better queuing performance than physically separated buffers, it requires carefully designed buffer management schemes for a fair and robust operation. This article presents a survey of the buffer management methods that have been proposed for shared-memory packet switches. Several buffer management policies are described, and their strengths and weaknesses are examined. The performances of various policies are evaluated using computer simulations. A comparison of the most important schemes is obtained with the help of the simulation results and the results provided in the literature. The survey concludes with a discussion of the possible future research areas related to shared-memory ATM switches.

Broadband ISDN networks, of which ATM is the accepted transfer mode solution, require fast packet switches to move the ATM cells along their respective virtual
paths. The prime purpose of an ATM switch is to route incoming cells (packets more generally) arriving on a particular input link to the output link, which is also called the output port, associated with the appropriate route [1]. Three basic techniques have been proposed to carry out the switching (routing) function: space-division, shared-medium, and shared-memory [2]. The basic example for a space-division switch is a crossbar switch, which has also served circuit-switched telephony networks for many years. The inputs and outputs in a crossbar switch are connected at switching points called crosspoints, resulting in a matrix type of structure. The operation of a shared-medium switch, on the other hand, is based on a common high-speed bus. Cells are launched from input links onto the bus in round-robin fashion, and each output link accepts cells that are destined to it. The subject of this article, the shared-memory (SM) switch, consists of a single dual-ported memory shared by all input and output lines. Packets arriving on all input lines are multiplexed into a single stream that is fed to the common memory for storage; inside the memory, packets are organized into separate output queues, one for each output line. Simultaneously, an output stream of packets is formed by retrieving packets from the output queues sequentially, one per queue; the output stream is then demultiplexed, and packets are transmitted on the output lines [2]. The block diagram of an SM ATM switch is depicted in Fig. 1. Examples of early shared-memory ATM switches are CNET's Prelude [3] and Hitachi's ATM switch [4].

Packet switches have another major functionality besides switching, namely, queuing. The need for queuing (also called buffering) arises since multiple cells arriving at the same time from different input lines may be destined for the same output port [2]. There are three possibilities for queuing in a packet switch: buffer cells at the input of the switch (input queuing); buffer at the output (output queuing); or buffer internally (shared-memory) [1]. Shared-memory ATM switches gained popularity among switch vendors due to the advantages they bring to both switching and queuing. In fact, both functions can be implemented together by controlling the memory read and write appropriately [5]. As in output buffered switches, SM switches do not suffer from the throughput degradation caused by
head of line (HOL) blocking, a phenomenon inherent in input buffered switches [1, 5]. Moreover, modifying the memory read/write control circuit makes the SM switch flexible enough to perform functions such as priority control and multicast [5]. Issues regarding the routing function of the SM architecture are outside the scope of this article. Likewise, we will not discuss the details of how the memory is organized into logical queues, and how the cells are written in and read out. Our focus is on the problem of buffer allocation. Buffer allocation determines how the total buffer space (memory) will be used by individual output ports of the switch. Our model of the SM switch is sketched from a queuing systems point of view, and is given in Fig. 2. The switch has $N$ output ports, and a total buffer space of $M$. A first-in-first-out (FIFO) buffer is allocated to each output port, denoted by $k_i$. The sum of the individual buffer allocations may or may not be larger than the size of the memory $M$.

A certain policy of buffer allocation is required in the SM switch to perform its queuing function. The selection and implementation of this policy is usually referred to as the buffer management. This article presents a survey of the research regarding the question: How should the total buffer space $M$ of the SM switch be managed to achieve better traffic performance (i.e., cell loss, cell delay)? Cell losses occur when a cell arrives at a switching node and finds the buffer full. Minimizing cell losses, or reducing them to acceptable levels, is extremely important to support any end-to-end application over networks.

This article is organized as follows. We present various buffer management methods, and simulation results are provided to compare the performance of the most important methods. We then present a comparative summary of the investigated buffer management schemes. Finally, we propose directions for future research.

**Buffer Allocation Policies**

Important examples of the early research on buffer sharing are presented in [68] and [9]. The common goal is the analysis of possible buffer allocation policies to achieve a
performance evaluation and comparison. The model in Fig. 2 is used, and the following stochastic assumptions are made in all the above cited papers:

- Packet arrivals are Poisson, i.e., they have exponentially distributed interarrival times (with rates $\lambda_i, i = 1, \ldots, N$).
- Packet lengths are exponentially distributed. Consequently, the service time for an output link is also an exponential random variable (with mean $1/\mu_i, i = 1, \ldots, N$).

The output link capacities are generally assumed to be equal, i.e., $\mu_1 = \mu_2 = \ldots = \mu_N = \mu$.

The class of policies that was studied in the late 1970s by Irland [6], and by Kamoun and Kleinrock [7], is called static threshold schemes in many contemporary publications, including [10]. We will use the same terminology, and start with the two simplest static threshold policies: complete sharing and complete partitioning. After static threshold schemes, we will study other approaches to buffer management, such as push-out and dynamic policies.

**Static Thresholds**

**Complete Partitioning and Complete Sharing** -- In the complete partitioning (CP) scheme, the entire buffer space is permanently partitioned among the N servers. The sum of the individual port buffer allocations is equal to the total memory M. Hence, CP actually does not provide any sharing. At the other extreme lies the second simple policy, complete sharing (CS). Here, an arriving packet is accepted if any space is available in the switch memory, independent of the server to which the packet is directed. In other words, individual buffer allocations equal the total memory space [7].

\[
CP: \sum_{i=1}^{N} k_i = M \quad (1)
\]

\[
CS : k_i = M, \ i = 1, \ldots, N \quad (2)
\]

where $k_i$ is the buffer allocated to port $i$, and M is the total buffer space. It is possible to make a comparison intuitively by looking at the definitions above. Under the CP policy, the buffer allocated to a port is wasted if that port is inactive, since it cannot be used by other possibly active links. On the other hand, under the CS policy, one of the ports may monopolize most of the storage space if it is highly utilized [8].

The assumptions of the traffic arrival process enable us to model the switch as a
Markov process. The state of the system is represented by a vector \( n = [n_1, n_2, \ldots, n_N] \) where \( n_i \) is the number of packets for the \( i \)th output port. The load of link \( i \) is defined as \( \lambda_i = \lambda_i / \mu \). It is shown in [6] that the steady-state probability \( P(n) \) is given in the following product form:

\[
P(n) = C \prod_{i=1}^{N} \pi_{i}^{n_{i}}
\]

where \( C \) is a normalizing constant chosen such that the sum of the probabilities of all states is unity. The space of the feasible states is determined by the policy that is chosen. The probability of any state outside the policy is zero. Figure 3 shows the state diagrams of CP and CS for \( N = 2 \) and \( M = 4 \). In the CS policy, a packet is lost when the common memory is full. In CP, a packet is lost when its corresponding queue has already reached its maximum allocation.

The assumption of exponential interarrival and service time distributions is not realistic for ATM systems. First of all, the fixed ATM cell size results in deterministic service times. Furthermore, the traffic in ATM networks is bursty in nature, implying a correlated traffic arrival process as opposed to the random traffic model presented in the earlier works [11]. A common way to model bursty traffic is by using two-state Markov chains. An on/off source, also named interrupted fluid process [11], is a two-state Markov process that is widely used for modeling ATM traffic sources. ATM cells are only generated during the on state with fixed interarrival time. The time spent in on and off states is exponentially distributed.

The model has three parameters: average length of the on period, average length of the off period, and the cell emission rate during the on period. The ratio of the on period to the sum of on and off periods is called the activity of the source, and is used as the main measure of burstiness, although there are various other measures [12]. Although analytical methods are considered in [13], the general method is to use computer simulations for the performance evaluation of SM queuing under bursty traffic [5]. We will also take this approach to compare some of the buffer management policies presented in this survey.

In the simulations, the service time of each port is normalized to 1. The traffic source parameters are selected as follows: mean duration of on state = 240; mean
duration of off state = 720; and cell interarrival time = 5. The destination port of the first cell in a burst (on period) is chosen randomly with uniform probability. The rest of the cells belonging to that particular burst are assumed to have the same destination as the first cell. The switch model has two output ports (N = 2): port 0 and port 1. The size of the shared memory is 300 cells (M = 300). The loads at the ports are increased by aggregating more sources. Two traffic conditions are investigated: balanced traffic, where the loads at the ports are equal; and imbalanced traffic, where the load at one port is varied, while it remains constant at the other port. Our performance metric is the cell loss ratio (CLR) at the ports. In balanced traffic conditions, we will only look at one port since the performance is identical at both ports.

Figure 4 provides a comparison of CP and CS under balanced traffic conditions. For medium traffic loads, CS achieves significantly lower CLR. In other words, a given CLR can be achieved with less buffer space in CS than in CP. This buffer reduction property of CS is one of the main reasons for the popularity of the SM switches.

The superior performance of CS with balanced traffic vanishes completely under imbalanced traffic. Figure 5 shows the CLR experienced at both ports while the load at port 1 is increased. The load at port 0 remains constant at 0.5. Here, both ports have the same CLR. The lightly loaded port 0 is being punished by the heavily loaded port 1. This, obviously, is an undesired behavior in a switch. The CP policy does not have this problem, as shown in Fig. 6. The port buffers are completely isolated in CP. Therefore, the lightly loaded port experiences a correspondingly low CLR, whereas the CLR at port 1 increases with the traffic load.

**Sharing with Maximum Queue Lengths** -- As shown in the works of Irland [6], and Kamoun and Kleinrock [7] for Poisson traffic, the two extreme policies CS and CP lead to undesirable behavior of the packet switch. The same result has been obtained for an ATM node under bursty traffic using simulations described in the previous section. In order to benefit from the efficiency of buffer sharing, but also to avoid the possible monopolization of the switch by one heavily loaded link, a "restricted buffer sharing" is proposed by Irland [6]. In this policy, a limit is imposed
on the number of buffers to be allocated at any time to any server. The sum of
these maxima is greater than the total memory space to provide sharing. This idea
is named sharing with maximum queue lengths (SMXQ) in [7]:

\[ \sum_{i=1}^{N} k_i \leq M, \]

where \( k_i \) is the size of the buffer allocated to port \( i \), and \( M \) is the size of the
memory. Although the above definition is more general, SMXQ is usually restricted
to the case where there is one global threshold for all the queues [57, 10]. With the
help of this restriction, all three policies (CS, CP and SMXQ) can be defined
through one threshold parameter \( \alpha \):

\[
\begin{align*}
CP: & \quad \alpha = 1/N \\
CS: & \quad \alpha = 1 \\
SMXQ: & \quad 1/N < \alpha < 1
\end{align*}
\]

The threshold parameter \( \alpha \) determines the degree of buffer sharing among ports.
An \( \alpha \) of \( 1/N \) corresponds to no sharing (CP), whereas an \( \alpha \) of 1 corresponds to full
sharing (CS). Any value in between is an SMXQ policy. The advantage of using
SMXQ is shown in Fig. 7. Here it is compared to CP, which had performed more
desirable than CS under imbalanced traffic in the previous section. It is seen that
SMXQ achieves lower CLR than CP, and at the same time manages to isolate the
"good" port from the "bad" port. The better CLR performance is obtained with
buffer sharing (does not exist in CP), and the isolation is obtained by restricting the
queue lengths (does not exist in CS). These comparative simulation results are in
agreement with previous work, such as [5].

In the particular example of Fig. 7, the threshold parameter is selected as 0.75;
hence, the queue lengths are restricted to 225 cells. Immediately the question
arises as to whether or not this particular \( \alpha \) value is the optimal one. The existence
of an optimal policy has been investigated in numerous works ([6, 9, 14, 15]). The
optimality metric is usually chosen to be the overall loss probability, or the loss
probability of a lightly loaded port. All the above cited papers use the Poisson
assumption for input traffic, which enables one to use analytical and computational
methods to determine the steady-state loss probabilities, and to find the conditions to minimize them. For more realistic traffic models (such as on/off sources), in relation to current computer networks, a similar optimal solution has not been formulated in the literature to our best knowledge. First of all, the loss performance of a single isolated buffer loaded with the on/off sources does not only depend on the mean traffic rate, but also on the burstiness and the mean burst length [16]. In addition, the interaction of queues in the SM environment appears to be much more complex under bursty traffic than it is under random traffic [13]. Consequently, the analysis studies and the investigation of optimum threshold values are currently performed with the aid of computer simulations [5, 10].

Figure 8 presents the effect of the threshold parameter on the loss performance for on/off sources. The curves give the CLR of the lightly loaded port 0 (load constant at $\lambda_0 = 0.6$) versus threshold parameter $\alpha$. The two curves correspond to two different loads at port 1 ($\lambda_1 = 0.9$ and $\lambda_1 = 1.1$). It is seen that the optimal value (the value that minimizes the CLR at port 0) of $\alpha$ is different in two cases. In the $\lambda_0 = 0.6, \lambda_1 = 0.9$ case the optimal $\alpha$ is approximately 0.625; in the $\lambda_0 = 0.6, \lambda_1 = 1.1$ case it lies around 0.55. It should be noted that the increase in port 1's already heavy load is shifting the policy toward CP, a result that is consistent with our previous observations.

**SMA and SMQMA** -- In [7] Kamoun and Kleinrock proposed two other policies: SMA (sharing with a minimum allocation) and SMQMA (sharing with a maximum queue and minimum allocation). In SMA a minimum number of buffers is always reserved for each port. SMQMA is the integration of SMA and SMXQ; each port always has access to a minimum allocated space, but they cannot have arbitrarily long queues. SMQMA has the following advantage over SMXQ. Although SMXQ restricts individual queue lengths, in a situation where there are many active ports, their total buffer occupation may leave a lightly loaded port with insufficient space. This is particularly important in current ATM networks, where there are different types of traffic classes with different priorities, and where connection admission control (CAC) plays a vital role to serve these different types of customers. (The issue of admission control will be handled in more detail in the last section.)
Therefore, in many shared-memory ATM switches, a minimum space is allocated for each port in order to simplify the issue of serving high-priority traffic in a buffer-sharing environment. For example, in Cisco's Lightstream 1010 switch, every port has a fixed reservation for high-priority continuous bit rate (CBR) traffic, whereas the rest of the buffer space is used with the SMXQ policy, resulting in an overall SMQMA operation [17].

**Push-Out**

The buffer sharing policies explained in the preceding sections have a common philosophy. An arriving packet is dropped at the instant of arrival if the switch is at a certain predetermined state in order to accept future arrivals from some other link which promises better throughput than the current arrival. However, there is always a chance that the decision to discard a packet to save space for another link may be a wrong one, and that the saved free space may not be used by other arrivals. In order to eliminate these situations, a delayed resolution policy (DRP) is proposed by Thareja and Agrawala in [14]. The DRP does not discard an arriving packet if there is space in the common buffer. If a packet arrives and the common buffer is full, the arriving packet, or some other packet that was already accepted, is discarded. The decision to drop a packet from a certain port can be made based on the state of the system or based on different priority classes. If the arriving packet is always dropped, then of course the policy is equivalent to CS. Wei et al. propose to drop from the longest queue in the switch, when the memory is full [18]. They call their algorithm drop-on-demand (DoD). This class of policies, in which a previously accepted packet can be dropped, is more commonly known as push-out (PO), and it has been studied with various different queuing systems. For example, push-out schemes have previously been used to provide service to multiple classes of traffic through one output buffer (and link) in an ATM switch [19]. A comparison of schemes in this type of buffer-sharing systems has been provided in [20]. In our context, where multiple output links compete for buffer space, the PO policy, as defined in [18], is appealing for the following reasons:
It is fair, as it allows smaller queues to increase at the expense of longer queues.
It is efficient, as no space is ever held idle while some queue desires more; thus, overall system throughput should be high.
It is naturally adaptive. When lots of queues are active, their rivalry keeps their queue lengths short; when only one queue is active, it is allowed to become long [10].

Figure 9 shows the performance of PO. It is compared to SMXQ in the same type of traffic conditions as the previous figure. It clearly achieves a lower CLR than the optimal SMXQ setting in both cases. This result is consistent with those reported in [10].

The drawback of PO is considered to be its practical implementation in an ATM switch. Although it has previously been stated that "the complexity of implementation of a DRP in a computer network is the same as that of the other policies," [14] PO seems to be difficult to implement in current fast ATM switches. In fact, according to Choudhury and Hahne [10]: "It is difficult to implement PO, however, in high-speed switches. When the SM is full, writing a cell into a queue involves the extra step of first pushing-out (in effect, reading out) another cell from a different queue, which could be any queue in the system. Moreover, PO requires that the switch monitor not only the individual queue lengths but the identity of the longest queue... Furthermore, it is fairly difficult to implement PO for traffic with multiple loss priority classes. Pushing out a low priority cell -- locating such a cell in the middle of a queue, excising the cell, then mending the break in the queue-- is not a trivial task."

Push-Out with Threshold -- The PO policy (or drop-on-demand) that is proposed in [18] treats each port equally. In ATM networks, where different traffic types have different quality of service (QoS) requirements, the resources in the networks should be appropriately allocated so the negotiated QoS requirements are satisfied. Hence, different ports carrying different traffic types might have different priorities. A modification to PO, complete sharing with virtual partition (CSVP), is proposed in [21] to achieve priorities among ports. A similar idea is proposed independently in [15], and is called push-out with threshold (POT). CSVP has the following attributes: N users (ports) share the total available buffer space M, which is virtually partitioned into N segments corresponding to the N ports, with the sizes $k_1, k_2, ..., k_N$, such that
When the buffer is not full, cells of any type are accepted upon arrival. When the buffer is full, there are two possibilities: if the arriving cell's type, for instance \( i \), occupies less space than its allocation \( k_i \), then, at least one other type must be occupying more than its own allocation, for instance \( k_j \). The admission policy will admit the newly arriving type \( i \) cell by pushing out a type \( j \) cell. If, on the other hand, the arriving cell's queue exceeds its allocation at the time of arrival, then the cell will be rejected.

When the buffer is not full, CSVP operates as CS. Under heavy traffic loads, the system tends to become a CP management. The partitioning of the buffer space \( M \) can be done based on the parameters negotiated at connection setup, or it can be estimated based on traffic monitoring or measurement. Thus, the buffer allocation does not need to be a static one, but can be adjusted dynamically to the traffic loading conditions [21]. The issues of changing traffic conditions and dynamic policies are handled in the next section.

**Dynamic Policies**

The analyses of the buffer allocation problem in all the studies cited above, with the exception of CSVP, assume static environments where traffic loads do not change with time. In most computer networks, however, traffic loads vary significantly with time. Changes in traffic characteristics can have several causes. The number of communicating users can change. The traffic routes in the network can change due to breakdowns or intentional alterations in routes. Further, the demands of users can change with time [14]. Hence, although a precise policy for managing the buffers to achieve optimum or near-optimum performance can be tailored for the specified loads, the performance can be far from optimum when the loads vary from their nominal values. We will mention two attempts to develop dynamic buffer management policies that can adapt to changes in traffic conditions.

Adaptive Control– In [14] Thareja and Agrawala approached the problem from an adaptive control systems perspective. Two key elements are given in the design of
an adaptive control system: identification and actuation. Identification refers to the measurement of the dynamic characteristics of the process to be controlled and the identification of the necessity for correction. Actuation signifies the generation of the actuating signals to modify the process behavior. In the buffer sharing problem, an adaptive policy should maintain an allocation that is the optimal policy for the given traffic load. Then, the identification process estimates the traffic arrival rates and determines when the current allocation should be revised. The actuation corresponds to enforcing the revised allocation. Since traffic estimations can be made by statistical measurements of the arriving traffic, and since the actuation merely involves enforcing a new buffer allocation, the key issue is the design of an updating procedure. Hence, the basic problem can be defined as follows [14]: "Given the values of $n(t)$ at time $t$, and $n(t_0)$ for some time $t_0$ when the allocation was last updated to match the optimal allocation, should the buffer allocation be updated at time $t$?"

A heuristic algorithm is proposed in [14] to solve the problem defined above. Although the basic idea of the adaptive control systems can be very valuable to develop new methods, the schemes in [14] have many shortcomings. For example, the Poisson assumption is considered unrealistic with current network traffic. This assumption and the restriction of only two output links makes possible the calculation of an optimal allocation for a given traffic load condition. Also, the statistical estimation of network traffic does not seem trivial today [12, 22], although it is considered to be a simple task in [14]. These might be the reasons why the optimizing control system approach has not gained much popularity among researchers. More recently, a different idea has been proposed by Choudhury and Hahne [10] to obtain a buffer management policy for ATM switches that can adapt to changing traffic conditions.

Dynamic Threshold -- The goal of [10] is to obtain a buffer management scheme that has the simplicity of SMXQ (called the static threshold (ST) scheme by the authors), and the adaptivity of PO. They aim to obtain this adaptivity without explicitly monitoring the traffic arrival to each port, as is done in [14]. Their scheme is called dynamic threshold (DT), and is based on the following idea: the queue
length thresholds of the ports, at any instant in time, are proportional to the current amount of unused buffering in the switch. Cell arrivals for an output port are blocked whenever the output port's queue length equals or exceeds the current threshold value. The DT scheme can be formulated as:

\[ T(t) = \alpha \cdot (M \cdot Q(t)) \]

(6)

where \( T(t) \) is the threshold for port buffers, \( \alpha \) is a constant, and \( Q(t) \) is the sum of all queue lengths. The DT scheme adapts to changes in traffic conditions. Whenever the load changes, the system will go through a transient state. For example, when a lightly loaded output port suddenly becomes very active, its queue will grow, the total buffer occupancy will increase, the control threshold will decrease, and queues exceeding the threshold will have their arrivals blocked temporarily while they drain, freeing up more cell buffers for the newly active queue. The major advantage of DT is shown to be its robustness to traffic load changes, a feature not present in ST policy. Figure 10 (which originally appeared in [10]) shows that setting the DT factor \( \alpha \) anywhere from 0.5 to 2.0 does a reasonably good job for all four balanced traffic conditions corresponding to sources with different mean burst lengths. However, for the same four traffic patterns, the optimal ST setting (given in maximum port queue lengths) ranges from 80 to 200, and there is no value that does a good job for all cases.

**Comparative Summary**

The class of static threshold (ST) policies, including CS, CP, SMXQ and their derivatives (SMA and SMQMA), have one important advantage: their ease of implementation. This is the main reason why, in fact, all commercially available ATM switches use ST policies. However, it has been shown that the performance of ST cannot be improved, and some possible undesirable situations cannot be prevented unless the threshold is tuned properly for the load conditions. Hence, the idea of monitoring traffic loads and updating the threshold parameter when necessary sounds appealing. However, the calculation of the optimal threshold value in a switch with many ports and probably many priority classes seems to be
a very difficult task. The observed complexity of current network traffic makes the optimization task even more difficult. The PO policy appears to be the answer for an adaptive scheme that does not require an updating procedure. It is also fair and achieves high throughput. However, PO has the disadvantage of being complicated to implement, because it involves the discarding of a cell that has already been accepted into the buffer. The DT policy, which is the newest of all, does not achieve the loss performance of PO. However, it is much more robust to load changes than ST, and it is much easier to implement than PO. A comparison of the policies is provided in Table 1. The adaptive control method of Thareja and Agrawala [14] is excluded because no comparative results are available. DT and ST are assumed to be operating with optimal threshold settings.

**Conclusions and Future Research**

The research in the field of SM packet switches that we tried to summarize in this article has shown the advantages of this architecture in terms of its queuing performance. Also, many buffer management policies have been proposed to get around its problems and shortcomings. Many switch vendors, including Lucent, Cisco and Hitachi, currently use SM switch fabrics in their products. The majority of the analyses in the literature involves connectionless networks, and remains within packet-level performance issues. More recent works in ATM's connection-oriented environment also focus on cell-level performance, disregarding performance metrics associated with connection-level issues. However, connection admission control (CAC) is an important part of the operation of ATM switches, and seems to be playing a vital role in future IP networks. CAC is a procedure that judges whether a new connection can be accepted to a particular switching node while still meeting the QoS requirements of the existing connections and the new connection. The number of admitted connections or the connection blocking probabilities are important metrics for the performance of a switching node. In an evaluation at the connection level, cells (or packets) cannot be treated as independent units since they belong to connections (or flows), which last much longer than the transmission
rate of a single cell. The work on CAC ([11, 23]) is generally based on single-port systems, and most CAC algorithms assume that the port has a fixed-size isolated buffer. Extending this approach to multiport shared-buffer systems presents a challenge for researchers. Work described in [24, 25] attempts to integrate existing CAC methods with memory-sharing. A more recent work ([26]) proposes a scheme called "call dynamic partitioning," which derives buffer allocation parameters from CAC parameters. It also supports different classes of services, and has the ability to handle unexpected input traffic deviations. The study of the interaction between different buffer management policies and different CAC methods appears to be an interesting area of future research.

Aside from the difficult problem of combining CAC with SM buffer management, another future research field may be the analysis of SM under self-similar traffic. Numerous measurements of real network traffic from different types of networks indicate that the traffic exhibits statistical self-similarity, or long range dependence [27]. The relevance of this statistical behavior to traffic engineering is an ongoing debate. As we saw the transition from random to bursty traffic sources in queuing analysis, we may see a new transition to self-similar sources. The cell- or connection-level performance evaluation of SM switches under self-similar traffic may have interesting results. Only one publication in this area is known to us [28]. The study of different service disciplines in connection with the SM buffering can also be an important area of future research. All of the work we presented so far is based on FIFO queues at the output ports. However, more complicated scheduling methods are found necessary to support the multiple QoS requirements of different users in ATM networks, and in future IP networks [29]. The development of a buffer management policy that works together with a desired service discipline would result in a more efficient allocation of the two critical network resources, buffer space and bandwidth.

References
1995, pp. 122939.


Additional Reading


Biographies

Mutlu Arpacı joined Georgia Tech in January 1997 as a Ph.D. student in electrical engineering. He is currently working as a research assistant in the Communications Systems Center at the Georgia Center for Advanced Telecommunications Technology (GCATT). Mutlu received a BS in electronics and telecommunications engineering from Istanbul Technical University in 1994. In 1996 he graduated from Louisiana Tech University with a MSEE degree. His current research interests are packet dropping mechanisms and active queue management in ATM and IP networks.
John A. Copeland holds the John H. Weitnauer, Jr., Technology Transfer Chair at the Georgia Tech School of Electrical and Computer Engineering. He is director of the Communications Systems Center, whose primary projects are related to the NSF vBNS (very-high-speed Backbone Network System), transmission of Internet protocol data over ATM switched networks, and the high-level design of a digital backbone for the State of Georgia. Prior to joining Georgia Tech in March 1993, Dr. Copeland was vice president, technology at Hayes Microcomputer Products (1985-1993), and vice president, engineering technology at Sangamo Weston, Inc. (1982-1985). Prior to that he was with Bell Labs (1965-1982). He has served on the Board of Trustees for the Georgia Tech Research Corporation (1983-1993). Dr. Copeland received B.S., M.S. and Ph.D. degrees in physics from the Georgia Institute of Technology. He has been awarded 41 patents and has published more than 50 technical papers. In 1970 he was awarded IEEE's Morris N. Liebmann Award for his work on gallium arsenide microwave devices. He is a Fellow of the IEEE and has served that organization as the Editor of the IEEE Transactions on Electron Devices.