Dynamic Bandwidth Allocation Policies

Yehuda Afek* Menashe Cohen† Eyal Haalman‡
Yishay Mansour§
Computer Science Department
Tel-Aviv University

September 22, 1995

Abstract

When traffic of connectionless best effort protocols such as IP is carried over connection oriented protocols with guaranteed bandwidth, such as CBR connection in ATM, the interface layer between the protocols (i.e., AAL - the ATM Adaption Layer) need to specify the bandwidth requirement and the duration of the bandwidth reservation. The purpose of this paper is to develop policies for deciding and for adjusting the amount of bandwidth requested for a best effort connection over such networks. Our aim is to develop such policies that achieve a good trade off between latency and utilization. The performances of the different policies are compared by an empirical evaluation.

1 Introduction

While connection oriented high speed networks (e.g. ATM) are expected to become the major networking infrastructure, most applications are still based on the Internet Protocol (IP). In order to lower entrance barriers for high speed networks, it would be necessary to carry IP connectionless traffic over the new generation ATM networks, and be compatible with IP applications.

The characteristics of the ATM and IP approaches are very different. IP is a connectionless best effort protocol, while ATM networks provide connection oriented with

*Telephone: +972-3-6408952, Fax: +972-3-6409357, e-mail: afek@math.tau.ac.il
†e-mail:menash@math.tau.ac.il
‡e-mail: eyalh@math.tau.ac.il
§Telephone: +972-3-6408829,Fax: +972-3-6409357, e-mail: mansour@math.tau.ac.il

guaranteed bandwidth services. That is, to carry an IP datagram in such networks a virtual circuit (VC) has to be setup with an indicated bandwidth requirement.

This mismatch in characteristics introduces several problems that the adaption layer has to resolve [SK, LPR]. For example, once a VC is open, and the datagram has been sent, the adaption layer has to decide how long to keep the VC open with the initial bandwidth. Clearly, if the rate of incoming packets matches the specified bandwidth allocation we would like to keep the VC open as it is. However, if packets arrive at a higher or lower rate, there is a need to change the specified bandwidth accordingly, or maybe even to close the VC.

Clearly, for any circuit with a specified bandwidth the accumulative requested bandwidth over the duration of the session is at least the total number of packets it has to deliver, because eventually all packets should be transmitted. Thus the average requested bandwidth should be at least the average packet rate. However, the bandwidth allocation could be either approximately the average of the traffic requirement, or much larger than it. Let us observe the two extreme cases:

On the one extreme, allocating sufficient bandwidth but just slightly more than the average of the traffic. Due to the bursty nature of the IP traffic, queuing would have to be used extensively, and long queues would occur during the duration of the VC, which would cause two major problems. First, queues may overflow which may result in the loss which in turn may cause retransmission of higher layer protocols (e.g. TCP). Secondly, the long queues cause the traffic to suffer high latency.

On the other extreme, a very large VC bandwidth allocation, say the peak bandwidth rate, would cause poor network utilization. If applications request much more bandwidth than they actually use the network resources will be under-utilized and other applications will be denied service. Furthermore, from the user point of view, assuming that the pricing policy depends on the bandwidth allocation of the VC, an over allocation of bandwidth implies that he or she are over charged.

For this reason, any bandwidth allocation policy should strive to maintain a balance between high network utilization and low latency, which is a classical trade-off in communication networks. This trade-off among these two parameters is at the heart of this research.

In this paper we introduce several bandwidth allocation policies for connectionless traffic over connection oriented guaranteed bandwidth networks[1], and compare them via simulation. We partition our policies into three groups.

1. Static algorithms in which the bandwidth allocation does not change.
2. Periodic algorithms in which the bandwidth allocation is adjusted periodically, in fixed equal size intervals and,

[1]Our settings most closely correspond to IP over CBR connections in an ATM network. ATM has defined and is devising standards and protocols for best effort services over the ABR connection which requires the specification of an upper bound on traffic requirements, called PCR and a lower bound called MCR.

3. Adaptive algorithms in which the bandwidth allocation may change whenever is necessary, as long as the changes are not too close to each other.

In this paper we assume that bandwidth allocation algorithms may modify the bandwidth allocation dynamically. A change in a VC bandwidth allocation is made by either closing the existing VC and opening a new one with the new allocation, or by changing the allocation of the VC without closing it, in case such an operation is supported by the network[2]. Both methods incur some overhead, thus these adjustments should not be made too often. However, as discussed above avoiding changes in a VC bandwidth allocation may cause either large queues and delays, or low network utilization.

The periodic and adaptive algorithms modify the bandwidth allocation dynamically. The main problem is in determining the new bandwidth allocation. If the future traffic is known then the problem is solvable in polynomial time (namely, finding the optimal latency for a given utilization level). However, the future traffic is unknown and packets arrive in an on-line manner. Therefore, it is necessary to predict future traffic as best as possible. Since it is impossible to change the bandwidth allocation too often, it is important that when a change is made, the new bandwidth allocation matches the future traffic as best as possible. To this end, forecasting methods are developed that predict the sessions arrivals rate in the near future (e.g. in the next period). (In our setting we are interested only in forecasting the total number of packets in the next period, which is of significant length, a task which is simpler than generating the actual traffic pattern.)

The development and comparison of the forecasting methods are the main contributions of this work. We have examined several online schemes to predict the future arrival rate. Out of which the best method was based on a three state machine Hidden Markov Model (HMM).

Formal definitions and notation are given in Section 2. The static and periodic algorithms are presented in Section 3 and the adaptive algorithms in Section 4. Conclusions can be found in Section 8. Theoretical analysis of the periodic algorithms is deferred to Section A in the Appendix.

1.1 related work

Saran and Keshav [SK] have introduced and studied the question of virtual circuit holding cost. That is, once a virtual circuit is open for say an IP application, how long should you hold the circuit open while no packets are arriving. However, they assumed that whenever a circuit is open its bandwidth is unlimited. Saran and Keshav have carried out an empirical evaluation and recommended an LRU-based scheme. Lund, Phillips and Reinglod [LPR] empirically studied the same problem and developed an adaptive scheme, which adapt to the inter-arrival time distributions of each circuit.

[2]Currently, ATM networks do not support such an operation.

2 Model and Problem statement

The smallest time units that we consider in this paper and have used in our simulation are called slots. Thus time is measured in slots (in the simulation a slot was 10^{-2} seconds).

Throughout the paper the term *session* may refer either to a session or to the time interval in which the session is alive. That is, from the time in which the first IP packet of the session arrives to the adaptation layer until the time that the last packet leaves the adaptation layer. The statement of the problem is as follows:

The input to our algorithms is the times (slots) in which packets of a session arrive to the adaptation layer. (In an online algorithm the input is only the number of packet arrivals in previous time slots.) The output expected by the algorithm is a specification of the amount of bandwidth requested and allocated for the session in the network in each time slot.

Our algorithms receive a parameter ρ, which affects the network utilization of the algorithm (and can be thought of as specifying the desired network utilization). The goal of the algorithm is *minimizing* the average queue length, for a given level of network utilization.

A trivial but impractical off-line solution would be to change the bandwidth at the beginning of each slot according to the number of packets that are expected to arrive during the time slot. Such a solution would attain good utilization with fairly minimal queue build up.

However it is not reasonable to assume that bandwidth allocation can be adjusted every time slot, in most cases we assume that a change in a VC bandwidth allocation is possible only once during each period of τ time slots. We will divide the *session* into periods where the k^{th} period begins in the beginning of the $(k-1)\cdot\tau+1$ time slot and ends in the end of the $k\cdot\tau$ time slot, $k = 1, 2, ..., n$. It is also not reasonable to assume that the algorithm receives information about the future arrivals. Thus, for most of our algorithms the input and the output of the problem will be as follows. At the end of each period (of τ time slots) the algorithm is given the arrival times of all the packets in previous periods. The output of the algorithm at this point of time is the requested bandwidth allocation for the session to be used in the next period.

We have designed and developed several algorithms to solve the problem as stated above. Most of the algorithms are allowed to adjust the bandwidth allocation only on period's boundaries. However we have also examined adaptive policies in which the bandwidth allocation may be changed at any point inside the period.

2.1 Testing

The algorithms have been tested with traffic patterns that have been collected using the "etherfind" program on our Ethernet network in Tel-Aviv University. Each pattern consists of 5000 Ethernet frames. Three different traffic patterns have been collected:

1. Traffic pattern between a host computer and an X-terminal.
2. Traffic pattern between a file-server and a host computer.
3. Traffic pattern between one host and another.

For each traffic pattern we tested each algorithm at various values of ρ. In each test the average queue length, $\hat{q}$, and actual utilization, $\hat{\rho}$, were measured, thus producing one data point, $(\hat{\rho}, \hat{q})$. All these points for one traffic pattern and one algorithm produce one graph such as for example in Figure 1. The x-axis in all the figures is utilization and the y-axis is queue length.

2.2 Notations

The following notations are assumed:

ρ - A parameter in the range $[0, 1]$ that represents the "desired utilization".

n - The number of periods in a session. Periods are numbered from 1 to n.

τ - The number of time slots in a period.

$T(t)$ - The number of packets arriving in time slot t.

$T_i(t)$ - The number of packets arriving in time slot t of period i.

T_i - The number of packets arriving in period i, i.e., $\sum_{t=1}^{\tau} T_i(t)$.

λ_i - The average packet arrival rate in period i (in packets per time slot), i.e., T_i/τ.

λ - The average packet arrival rate over the entire session, i.e., $\sum_{i=1}^{n} \lambda_i/n$.

μ - The bandwidth allocation for the entire session (used for the static algorithms only).

μ_i - The bandwidth allocation in period i (in packets per time slot).

$\hat{\rho}$ - The average session utilization, i.e., $\frac{\sum_{i=1}^{n} T_i}{\sum_{i=1}^{n} \tau\mu_i} = \frac{\sum_{i=1}^{n} \lambda_i}{\sum_{i=1}^{n} \mu_i}$.

q_i - The queue length at the beginning of period i.

$q_i(t)$ - The queue length in time slot t of period i.

$\hat{q}_i$ - The average queue length in period i, i.e., $\sum_{t=1}^{\tau} q_i(t)/\tau$.

3 The algorithms

We can classify the algorithms that we considered into three general categories:

Static algorithms - open a VC and fix its bandwidth once at the beginning of a session and do not modify the allocation during the session.

Periodic algorithms - may adjust the bandwidth allocation of a session every period of τ time slots, at the beginning of the period. The output of the algorithms is the requested bandwidth for each period.

Adaptive algorithms - which are more flexible than the periodic algorithms. Here the bandwidth of a session may be adjusted at arbitrary times. They will of course, have some limitation on the number of changes performed and their frequency.

3.1 Static algorithms

We consider the following off-line static algorithm `stat`, which opens the session with bandwidth, $\mu = \frac{\lambda}{\rho}$. Although this is an off-line algorithm (it has to know the average packet arrival rate in advance) we chose `stat` as a representative of the static algorithms since clearly no static algorithm can do better than `stat`. As we shall see later, static algorithms, and `stat` in particular, have poor performances.

3.2 Periodic algorithms

Any periodic algorithm selects at the beginning of each period i the bandwidth allocation μ_i for the duration of period i. One set of on-line periodic algorithms that we consider use a combination of q_i, the number of packets in the queue at the beginning of period i, and λ_{i-1} the average arrival rate in the previous period, to make the selection at the beginning of period i. The unrealistic off-line version may also use λ_i the average arrival rate in the next period. The various periodic algorithms compute μ_i as follows:

`prev` - $\mu_i = \frac{\lambda_{i-1}}{\rho}$ for $i = 1, \ldots n$, and $\lambda_0 = 0$. That is, allocate for the next period the amount of bandwidth necessary to transmit the same amount of traffic as has arrived in the previous period in the desired utilization.

`queue` - $\mu_i = \frac{\frac{q_i}{\tau}}{\rho}$. That is, allocate for the next period the amount of bandwidth necessary to transmit the current content of the queue in the desired utilization.

`queue+prev` - $\mu_i = \frac{\frac{q_i}{\tau} + \lambda_{i-1}}{\rho}$ for $i = 1, \ldots n$, and $\lambda_0 = 0$. That is, allocate for the next period the amount of bandwidth necessary to transmit the current content of the queue plus the same amount of traffic as has arrived in the previous period, in the desired utilization.

`queue+next` - $\mu_i = \frac{\frac{q_i}{\tau}+\lambda_i}{\rho}$, $\lambda_0 = 0$ (an off-line strategy). That is, allocate for the next period the amount of bandwidth necessary to transmit the current content of the queue plus the amount of packets that will arrive in the next period, in the desired utilization.

Two basic properties give the periodic algorithms a clear advantage over the static algorithms (even the off-line static algorithm `stat`), when the period duration is sufficiently small:

1. The bandwidth can be allocated in each period such that erroneous allocations in previous periods may be compensated for in future periods.
2. The traffic in one period is more homogeneous and may be effectively estimated by the traffic of previous periods.

The first algorithm, `prev`, can be intuitively described as follows: In each period we allocate the amount of bandwidth necessary to transmit all the packets received in the previous period with a certain utilization. That is, we allocate a certain bandwidth that if all that we transmit in that period is $T_{i-1} = \tau\lambda_{i-1}$ then the connection would be idle for a fraction of $1-\rho$ of the time. However, due to the fluctuations in traffic, in some periods the actual utilization will be more than ρ while in others it will be less than ρ.

In `queue` we allocate sufficient bandwidth to transmit all the packets in the queue at the end of the previous period, and have utilization ρ. Note `queue` is very similar to `prev`, however they are different. In `prev` even if most of the packets of the last period have been already transmitted, it still allocates enough bandwidth to transmit an equal amount in the next period. On the other hand `queue` would allocate a smaller amount of bandwidth in such a case. Note that in this case ρ is just an approximation of the desired utilization. In all our experiments we use ρ as a parameter to the algorithm and later measure the actual utilization, $\hat{\rho}$, this later value is the one we use in our plots. Further note that for $\rho = 1$ `queue` and `prev` are exactly the same.

The next algorithm, `queue+prev` which following the above discussion makes the most sense, allocates for the next period enough bandwidth to transmit the packets that are in the queue now plus an amount of traffic that is equal to what has arrived in the previous period. Notice that for a fixed ρ, `queue+prev` allocates always at least as much as `prev`. Again, we use ρ only as a parameter that influences the utilization of the algorithm, later we measure explicitly the actual utilization $\hat{\rho}$.

In the appendix we prove several theoretical bounds on the average and worst case queue length in some of the schemes. Let us here mention the results and the interested reader may find the proofs in the Appendix. We show that allocating bandwidth sufficient to empty the queue (i.e., `queue` with $\rho = 1$) gives a simple upper bound on the queue length in the session. More precisely, algorithm `queue` with $\rho = 1$ will never have a queue larger than twice the maximum traffic arriving within a period. The idea is that at each period we transmit the packets arriving in the

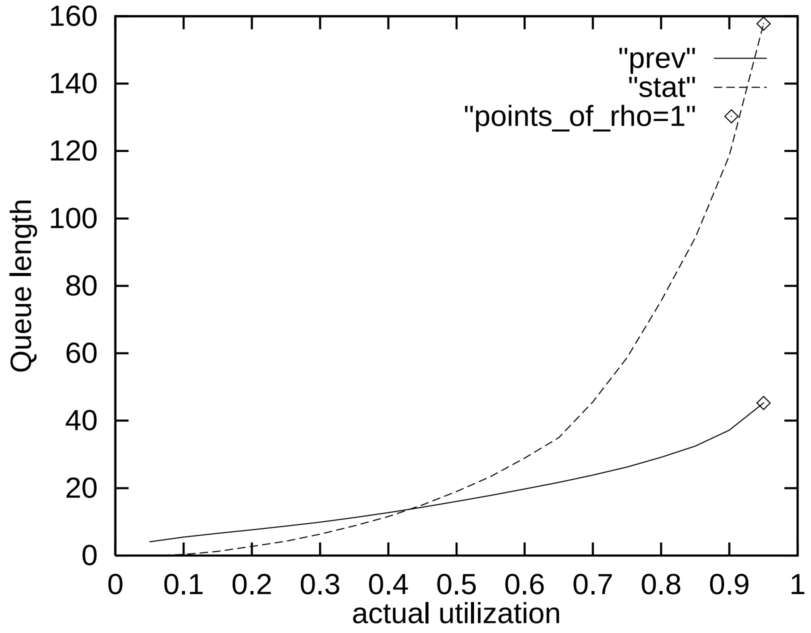


Figure 1: A comparison of `stat` and `prev`.

current and preceding period, therefore, in the worst case, a packet may be in the queue in two consecutive periods (the one that it was generated in and the following one, in which it was delivered). We also show that this bound is indeed tight.

The above argument bounded the worst case queue length by twice $T_{max} = \max_{i=1}^{n} T_i$. We show that the average queue length is bounded by $\frac{3}{2}(T_{avrg} = \frac{1}{n}\sum_{i=1}^{n} T_i)$, and show that this bound is tight.

Finally, we show upper bounds on the queue length when using `queue+next`, with $\rho = 1$. For the worst case, the same arguments as for `queue` bound the maximal queue length in the session by the maximum traffic arriving in two consecutive periods and show that this upper bound is tight. The interesting phenomena occurs for the average queue length. For this case we are able to show a bound of $\frac{5}{8}\sum_{i=1}^{n} T_i/n$, which is significantly better than the bound for `queue`.

In Figure 1[3] we can see that for $\hat{\rho} \geq 0.4$ the average queue length in `prev` is smaller than in `stat` and the difference becomes more significant for larger values of $\hat{\rho}$. For smaller $\hat{\rho}$ `stat` is slightly better, because in `stat` a VC is always open where in `prev` a burst might arrive when the VC is closed. This could happen if for example $T_i = 0$ so `prev` will allocate $\mu_{i+1} = 0$, but in the beginning of period $i+1$ a burst might arrive, which causes the queue to build until the next period. Still, this case does not justify low utilization. Moreover `stat` is an off-line algorithm where `prev` is an on-line algorithm.

[3]In this figure, like all the figures we present, ρ can not be greater than 1, hence the utilization not necessary attain a value of 1.

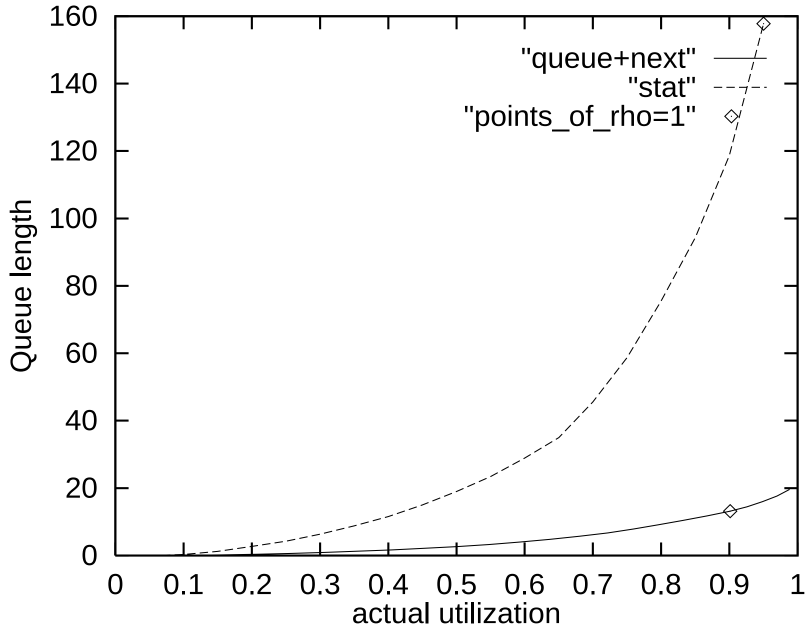


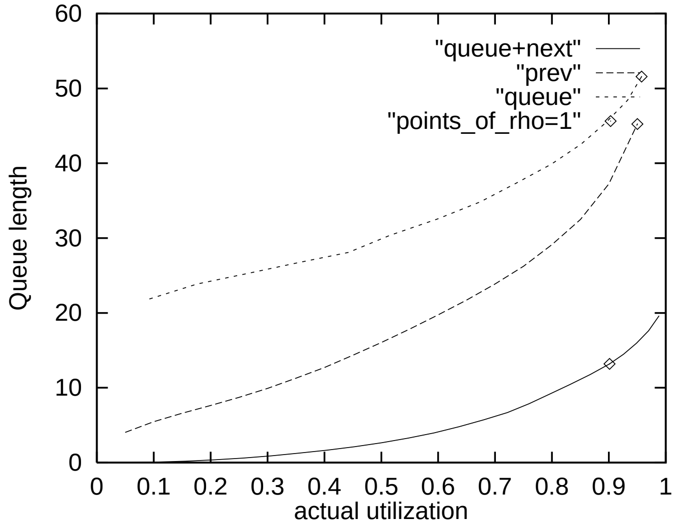
Figure 2: A comparison of `stat` and `queue+next`.

In Figure 2 we can see a comparison between `stat` and the off-line algorithm `queue+next` which allocates for the next period bandwidth proportional to emptying the current queue plus the arrival rate in the next period. In this graph it is clear that `queue+next` is better for any level of utilization $\hat{\rho}$. This leads us to the conclusion that dynamic algorithms are better and should be used as long as we do not change the bandwidth allocation too often. Thus, from now on we will consider only the dynamic algorithms and compare among them.

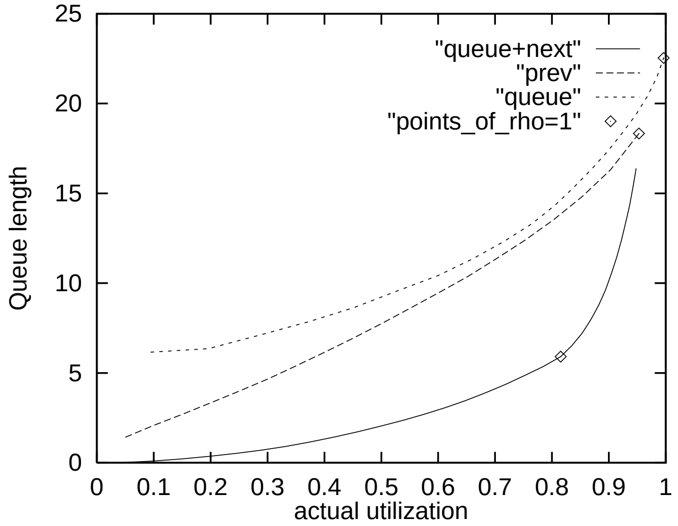
Figure 3 has a comparison between the three introduced periodic algorithms on different communication patterns. The graphs strength our expectations and the theoretical analysis that `queue+next` gives the best results among the three periodic algorithms, in all the different scenarios. Unfortunately `queue+next` is an off-line algorithm and thus we should search for an equivalent on-line algorithm. The additional information that it has is the access to the number of packets arriving in the next period. In order to adapt it to an on-line algorithm we need a way to forecast the packet arrival in the next period. Based on the heuristic that the network arrival in period i is similar to that of period $i-1$, i.e., $\lambda_i \approx \lambda_{i-1}$, we derive algorithm, `queue+prev`: in which $\mu_i = \frac{\frac{q_i}{\tau}+\lambda_{i-1}}{\rho}$, $1 \leq i \leq n$, $\lambda_0 = 0$.

In Figure 4 we can see a comparison between `queue+prev` and the other periodic algorithms. We can see that `queue+prev` achieves better results than `queue` or `prev`, but still falls short of `queue+next`.

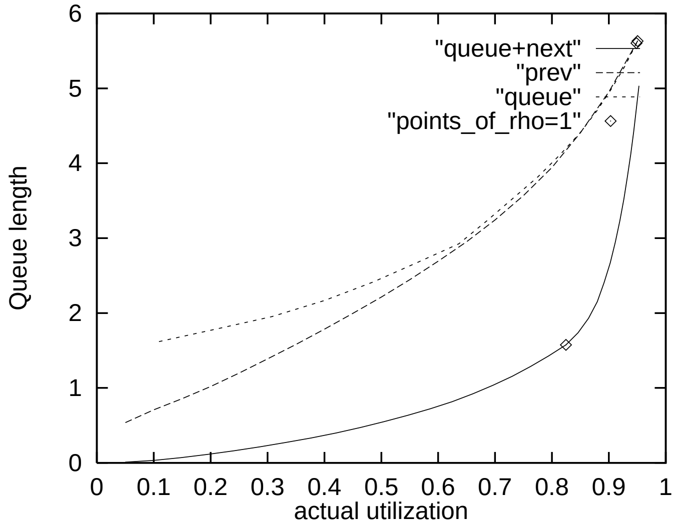
Based on the assumption that the packet arrival in period i is similar to the packet arrival in the period $i-1$, we tried several algorithms in which we divided the last period into sub periods where each sub period received a different weight. The algorithms that gave a slightly higher weight to sub periods at the end of the period



a



b



c

Figure 3:Fig a - file server to host.Fig b - host to host.Fig c - x-terminal to host.

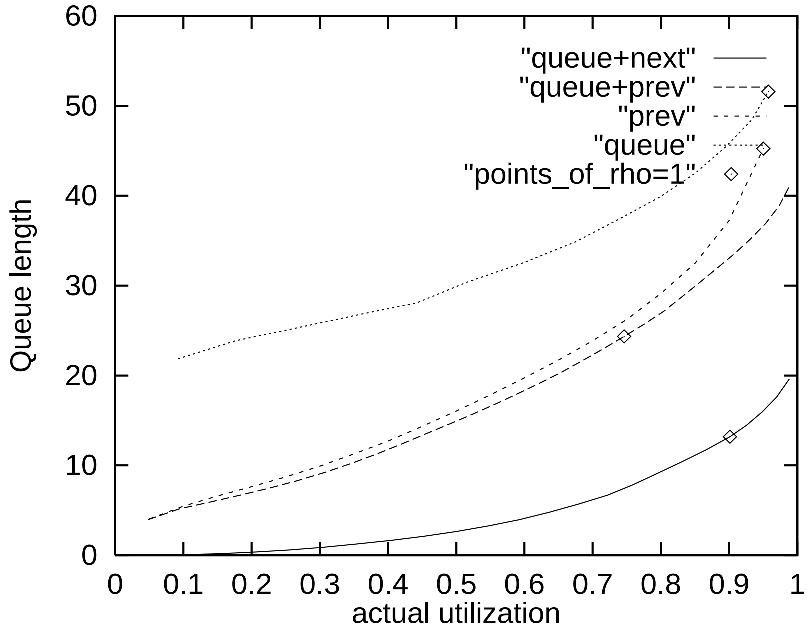


Figure 4: A comparison of `queue+prev`, `prev`, `queue` and `queue+next`.

gave the best results even though they where not much better than `prev`.

As an example to illustrate the result we use 10 sub periods. The weight given to the ith sub period was proportional to $(1.1)^i$, i.e., $\frac{(1.1)^i}{\sum_{j=1}^{10}(1.1)^j}$. The result of this algorithm where only slightly better than those of `prev`. This can be seen in Figure 5.

3.3 Hidden Markov Model (HMM)

In many network applications it is customary to abstract the network traffic as a three state machine with: (1) an idle state, where no packets arrive, (2) an active state, where packets arrive in a slow rate and (3) a burst state, where packets arrive at a high rate. An example which illustrates this intuition may be a file transfer. Initially the system is idle, then the user initiates the transfer (active state), finally the file transfer is actually performed (burst state). Another example is an interactive conversation using talk, usually one side writes (active state) and the other reads (idle state) and vice versa.

In a Hidden Markov Model (HMM) we can attach to each state a random variable that measures the number of packets generated in a time slot, i.e., $T_i(t)$ is distributed according to the state of the system at time t. The system changes its state in a stochastic way, i.e., there is a conditional probability distribution of the next state given the current state.

An HMM is a doubly stochastic process with an underlying hidden stochastic process. The states of the underlying stochastic process behavior can not be observed

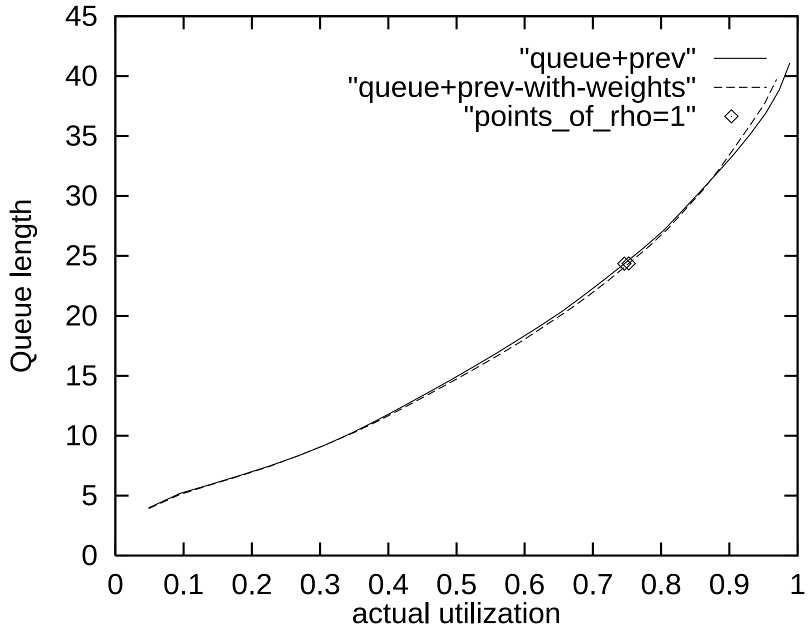


Figure 5: Comparing `prev` performance using sub periods.

directly. What can be observed are the outputs (in our case the number of packets generated). Our aim is to use HMM to forecast the incoming traffic in the next period.

For our experiments we need to predict the number of packets arriving during the next period. In order to do that, we "learn" a state machine that describes best the behavior of the network in the recent past (several previous periods) using the learning algorithm of [JR]. (In a nutshell, the algorithm performs an Expectation Maximization (EM). In each iteration, the maximum likelihood partition into states is computed, assuming that the current model that we have is the correct one. Then, given the partition that we computed, the parameters of the model are estimated.)

There is a strong relationship between the network behavior and our ability to make "useful" forecasts. Denote by $R(s)$ the random variable which is the number of packets generated during a period, given that we start at state s. In order for the model to be beneficial for our purposes, the expected value of $R(s)$ for different states s should be significantly different (otherwise we will make the same allocation independent of the state that we are in). In addition, the variance of $R(s)$ should be small (otherwise our estimate would be highly noisy). In fact there is a relationship between how "mixing" the Markov model is and how large can we chose the period, i.e., the value of τ.

In all the packet arrival patterns we have tried, the HMM algorithm achieved the best or nearly the best results compared to all the other on-line algorithms. In Figure 6 we can see a comparison between HMM, `queue+prev` and `queue+next`.

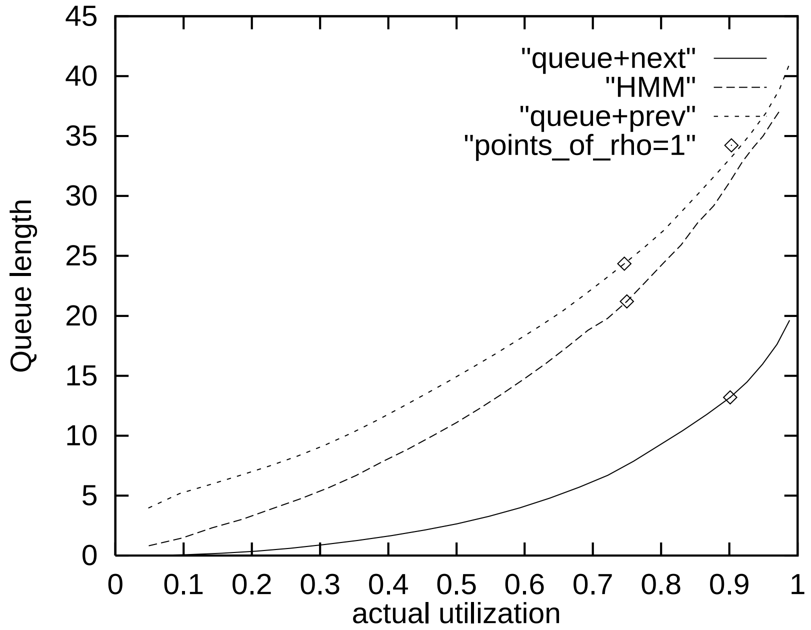


Figure 6: A comparison of HMM, `queue+prev` and `queue+next`.

3.4 Neural networks

Another method we tried to use for forecasting the number of packets arriving in the next period is neural networks. In this experiment we did not assume any structure of the data, we trained the neural network on several files of data (used as training sample set) and tested the obtained neural network on different files (used as testing sample set). Our motivation was to try another popular learning tool and see how well it does.

Our input to the neural network was two consecutive periods of packet arrival patterns (i.e., for each time slot in the last two periods the number of packets that arrived), and the output was the number of packets generated in the next period.

The results show that the average queue length generated by the neural net algorithm was worse than `queue+prev` and significantly worse than HMM. This can be seen in Figure 7.

4 Adaptive algorithms

4.1 Simple adaptive algorithm

The idea behind this adaptive algorithms is rather simple. In the periodic algorithms we made a change in the bandwidth allocation at the end of each period. In this algorithm, `simple adaptive`, we will try to avoid some of the changes, when the change is insignificant. In order to see whether a change is significant, either the

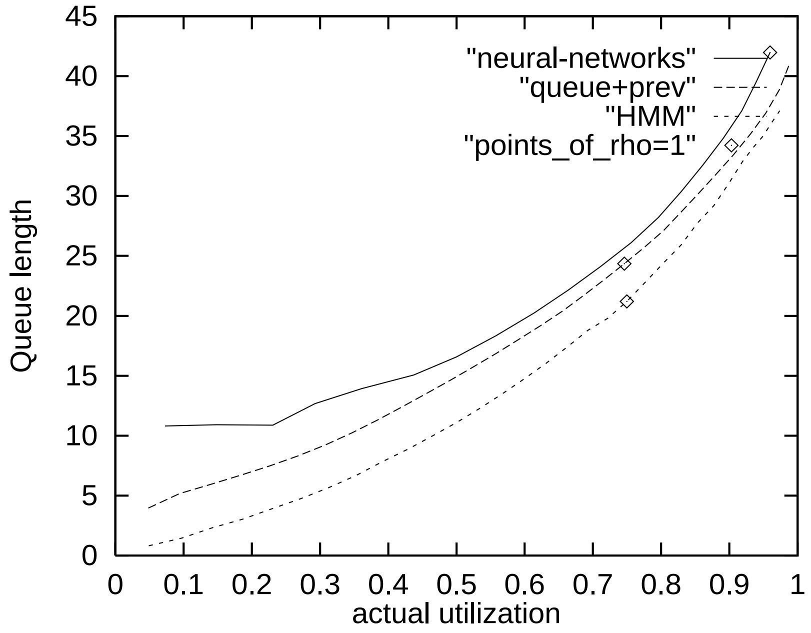


Figure 7: Neural network results

relative change or the absolute change has to be significant. The idea is to see whether we can save many bandwidth modifications without affecting the performance of the algorithm.

`simple adaptive` has a parameter, $CHANGE$, where $0 < CHANGE \leq 1$. Like all the periodic algorithms we calculated in the beginning of period i the desired μ_i which will be denoted by $\mu_{i_{new}}$. Now μ_i is determined as follows:

IF $(CHANGE \leq \mu_{i-1}/\mu_{i_{new}} \leq 1/CHANGE)$ THEN

$$\mu_i = \mu_{i-1}$$

ELSE

$$\mu_i = \mu_{i_{new}}$$

Note that for CHANGE=1 this adaptive algorithm behaves exactly as the periodic ones do and for $CHANGE$ close to 0 it behaves like the static ones. For $CHANGE < 1$,`simple adaptive`, unlike the periodic ones, does not change the bandwidth allocation if the desired bandwidth allocation is not much different than the current one. Thus, it can reduce the number of bandwidth allocation changes. As expected for smaller CHANGE the number of bandwidth allocation changes becomes smaller and the results become worse. However, the overall affect of this adaptive technique seems to be negligible. For example at 95% utilization, there was a saving of 10% of the bandwidth modification at the cost of 5% increase in the average queue length.

4.2 Adaptive HMM

When we used the HMM to forecast the traffic in the next period, a crucial element in the prediction was the HMM state in the beginning of that period. As long as the HMM remained in the above state, our estimations were quite accurate, but occasionally the HMM changed it state much sooner than we expected, sometimes right at the beginning of the next period - resulting in bad time prediction and thus low utilization or high queue length.

A trivial solution might be to disregard the periods and change the bandwidth allocation every time the HMM changes it state. The problem with this solution is that when we do change the bandwidth allocation we wish to empty the current queue. However, In order to this, we need to know when will be the next change.

The following algorithm `adaptive HMM` solves the above problem. This algorithm learns the HMM in the same way HMM does. However, in addition to changing the bandwidth allocation every period, it may change it as well every time the state machine changes its state. The allocated bandwidth is just enough to empty the current queue length and transmit the expected traffic by the end of the period.

This algorithm gives very good results, as can be seen in Figure 8. however, it changes the bandwidth allocation more than the periodic algorithms with the same period length do. (in several experiments we performed the number of bandwidth allocation changes increased within a factor of up to 2. In Figure 8 this number increased within a factor of 1.7) A great deal of those changes took place close to the end of the period. In such cases it could be wiser not to change the bandwidth allocation till the end of the period in order to reduce the number of bandwidth allocation changes.

In Figure 9 there is a comparison of `adaptive HMM` with $\tau = 300$ and other algorithms with $\tau = 250$. `adaptive HMM` was modified so it changes the bandwidth allocation only if the state transition occurs within the first half of the period and only if it is the first change within this period. In Figure 9 all algorithms change the bandwidth allocation exactly the same number of times.

4.3 Off-line Adaptive

We have already introduced `queue+next` which is a periodic off-line algorithm. The algorithm `queue+next` achieved good results, but still it is not the optimum. The following is also an off-line algorithm, but it can change the bandwidth allocation in every time slot in the period. Still the changes in bandwidth allocation can be made only once in a period. The period intervals are fixed as before. This algorithm is the closest to the optimum that we have implemented. We did not try the optimum off-line algorithm because its testing would require an unrealistic amount of space and time. (Although the optimal solution may be computed in polynomial time, since it is based on dynamic programming, the cubic running time made it infeasible for the parameters we were interested in.)

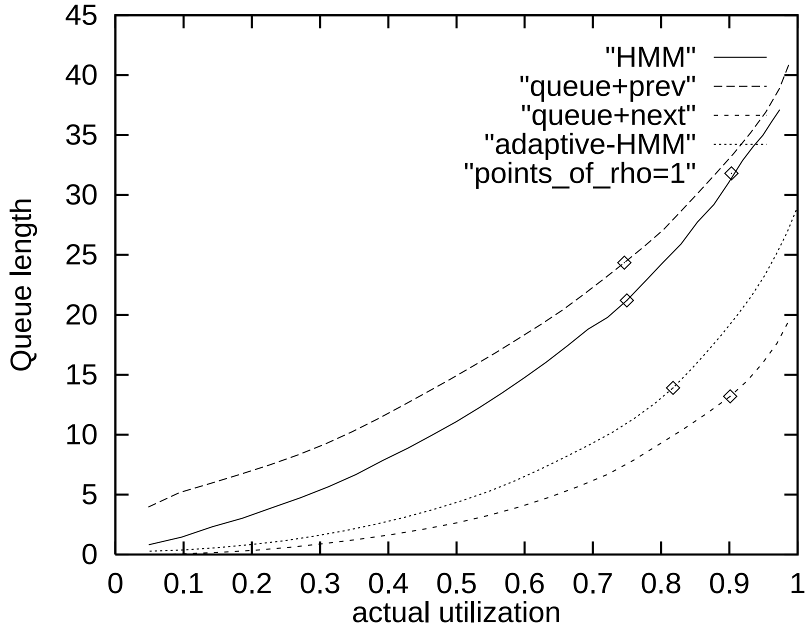


Figure 8: Adaptive HMM vs other algorithms - same period length

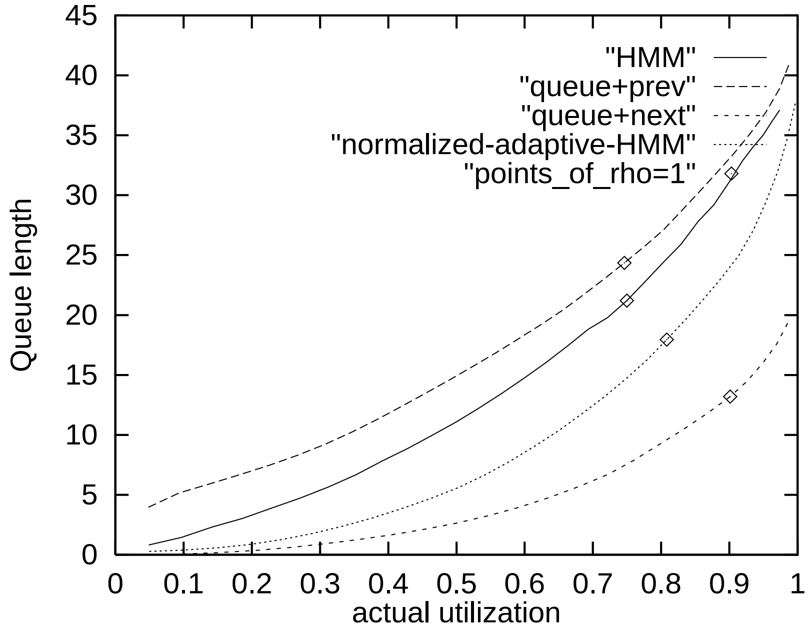


Figure 9: Adaptive HMM vs other algorithms - same number of bandwidth allocation changes

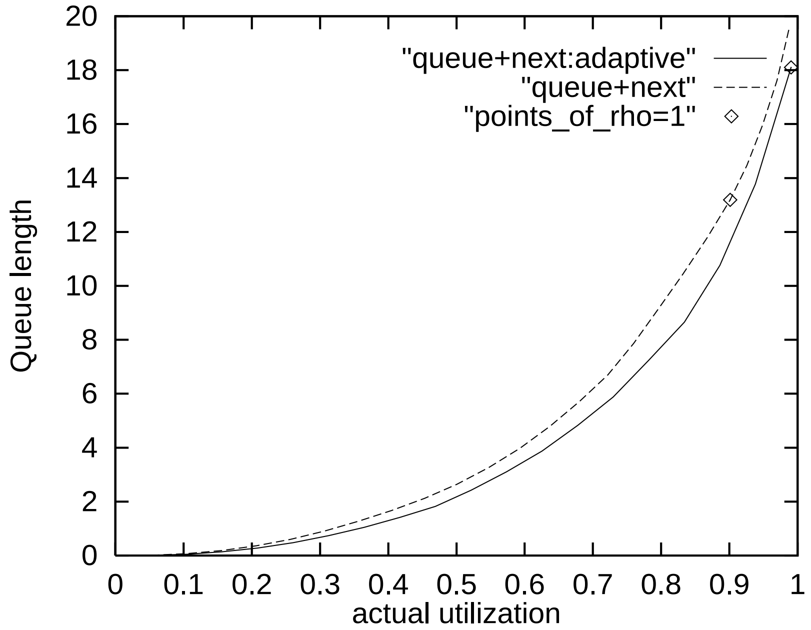


Figure 10: A comparison of `queue+next` and `adaptive queue+next`.

Formally the algorithm is defined as follows: denote $\mu_{i_{adapt}}$ as the bandwidth allocation in the end of period i. Denote t as the time slot to change the bandwidth allocation, t is measured by the number of time slots past from the beginning of the period ($0 \leq t \leq \tau$). Given the above t and requiring that the utilization will be exactly ρ the following should hold:

$$T_i/\rho = t \cdot \mu_{i-1_{adapt}} + (\tau - t) \cdot \mu_{i_{adapt}}$$

and we will obtain that:

$$\mu_{i_{adapt}} = ((\lambda_i \cdot \tau/\rho) - \mu_{i-1_{adapt}} \cdot t)/(\tau - t)$$

The optimal t is determined according to the minimum average queue length for the current period. It is determined independently of the optimal t of other periods to obtain a reasonable run time. Thus, theoretically, in some cases `queue+next` can achieve better results.

In Figure 10 we can see a comparison between `queue+next` and this adaptive algorithm. We can see that the adaptive algorithm gives only slightly better results. In other cases the two algorithms gave similar results.

5 $\hat{\rho}$ vs ρ

A part from deciding which is the best algorithm for a desired network utilization, $\hat{\rho}$, the interface layer should also ran the chosen algorithm with a suitable ρ in order to achieve the above network utilization.

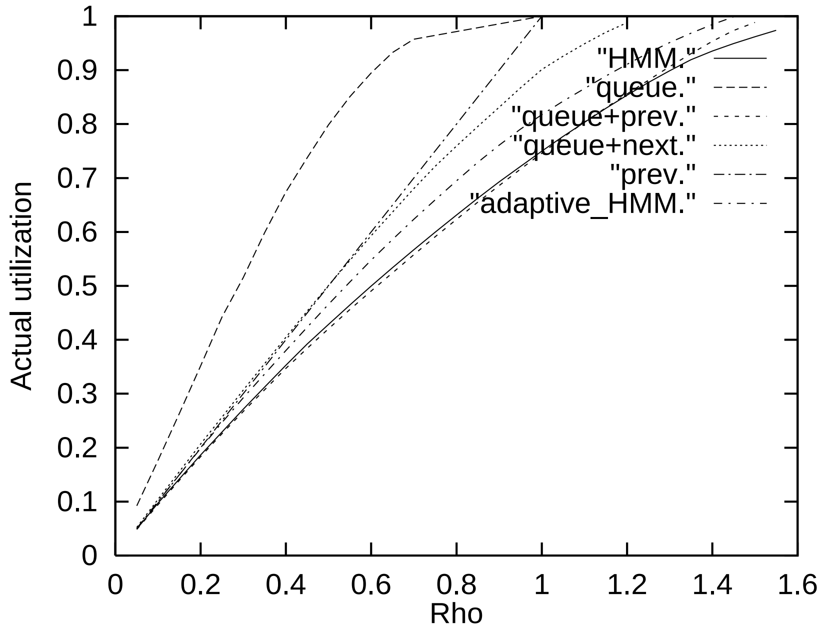


Figure 11: Actual utilization vs. rho

While several algorithms, like `stat` or `prev`, maintain $\hat{\rho} = \rho$ other algorithms have a bit less trivial relation between the two.

Figure 11 shows $\hat{\rho}$ as a function of ρ for several algorithms. All the algorithms in Figure 11 apart from `prev` and `queue` allocates bandwidth sufficient to empty the current queue and an estimation of the future bandwidth. Even when this estimation is precise , e.g. `queue+next`, the actual utilization is worth than ρ as the packets in the next period can arrive near the end of that period. moreover when the estimation is based upon the previous period, e.g. `queue+prev`.

The only algorithm where the utilization is better than ρ is `queue`. The reason for that is that when $\rho < 1$ the packets that arrive in the current period are also transfered, and unlike `prev` those packets do not affect the bandwidth allocation of the next period.

6 $\hat{q}$ vs τ

The theoretical results we achieved show that the average queue length in a session is bounded by a constant time the average bandwidth in a session.

This magnitude, T_{avrg}, in turn is linearly depended on the period length. Thus, grossomodo, the longer the period length is the longer we expect the average queue length to be. However, for a specific session and a specific algorithm, we can expect some local minimums.

Figure 12 and Figure 13 show $\hat{q}$ as a function of τ.

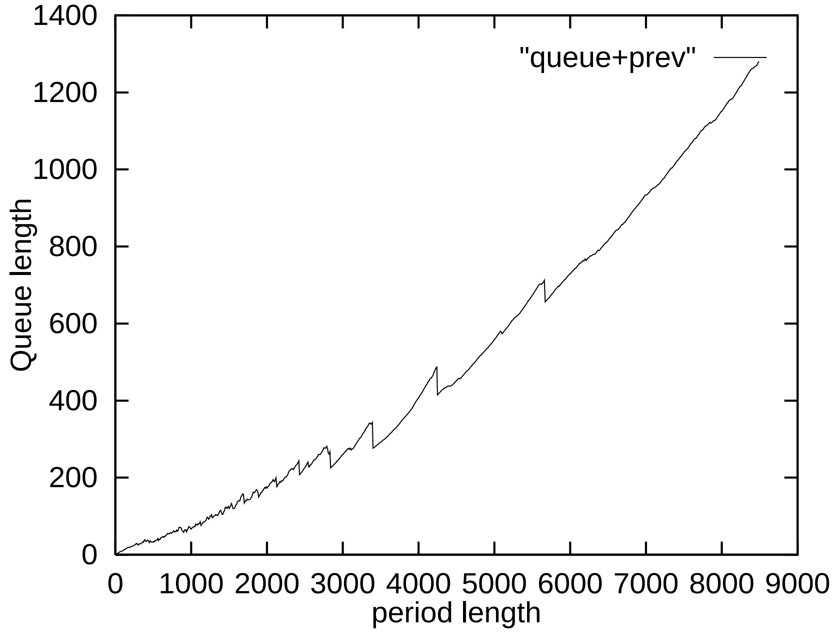


Figure 12: Average queue length vs. period length - `queue+prev`

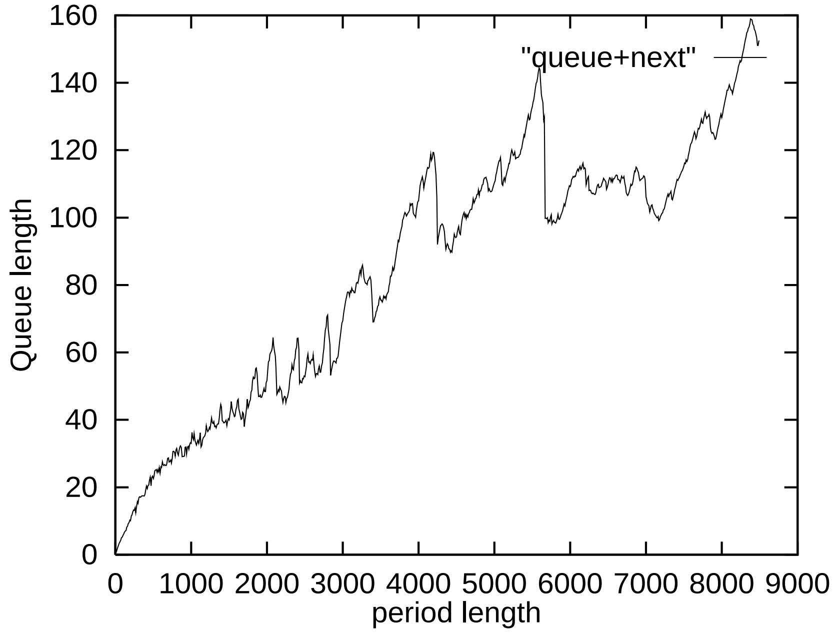


Figure 13: Average queue length vs. period length - `queue+next`

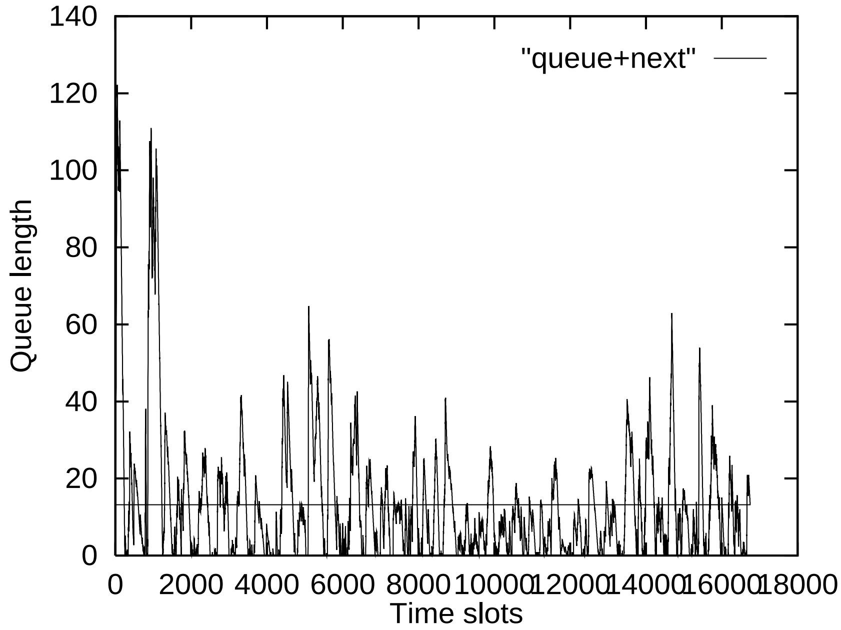


Figure 14: The queue length of `queue+next` for $\rho = 1$.

7 queue length as a function of time

It is interesting to see the queue length all over the session and not only the average queue length of the session. In figures 14 , 15 we can see the appropriate graphs for that. The queue length in the first quarter of the session with marking of the a start of a period can be seen in figures 16 , 17

8 Conclusions

There are several conclusion one can draw from this work. First, dynamic algorithms, that may change the bandwidth allocation periodically, have a clear advantage over static algorithms, even if the static algorithms are off-line static algorithms.

The simple heuristic that the near future is very similar to the near past ("what has happened is what will happen") gives fair results. That is, the simple dynamic algorithm that allocates at the beginning of each period enough bandwidth to empty the current queue plus to transmit the same amount of traffic as has arrived in the last period, is both simple and gives fair results.

Finally, the technique of HMM is very well suited for the problem that we were considering, and seems to be the winner in all the different scenarios that we investigated. One needs to remember that there is a computational burden in using the HMM learning algorithm in an online manner.

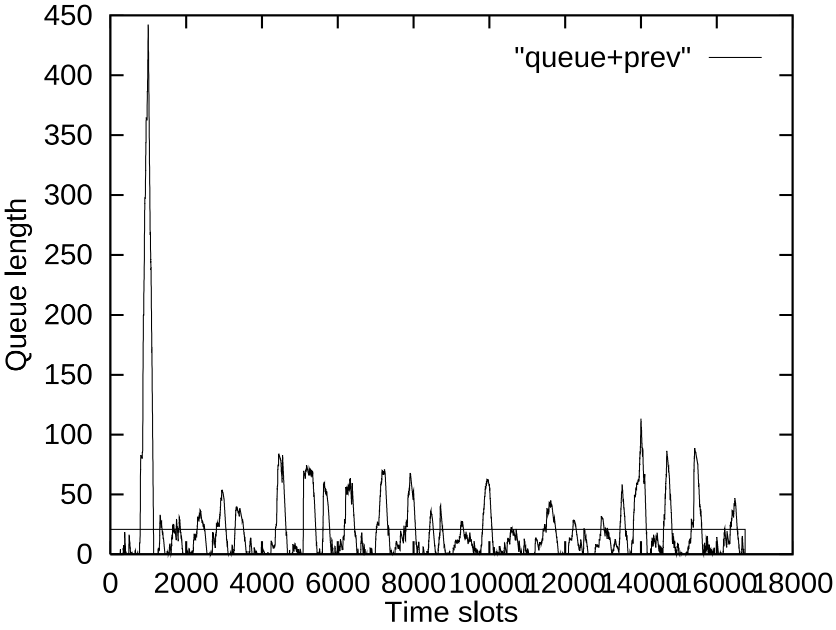


Figure 15: The queue length of `queue+prev` for $\rho = 1$.

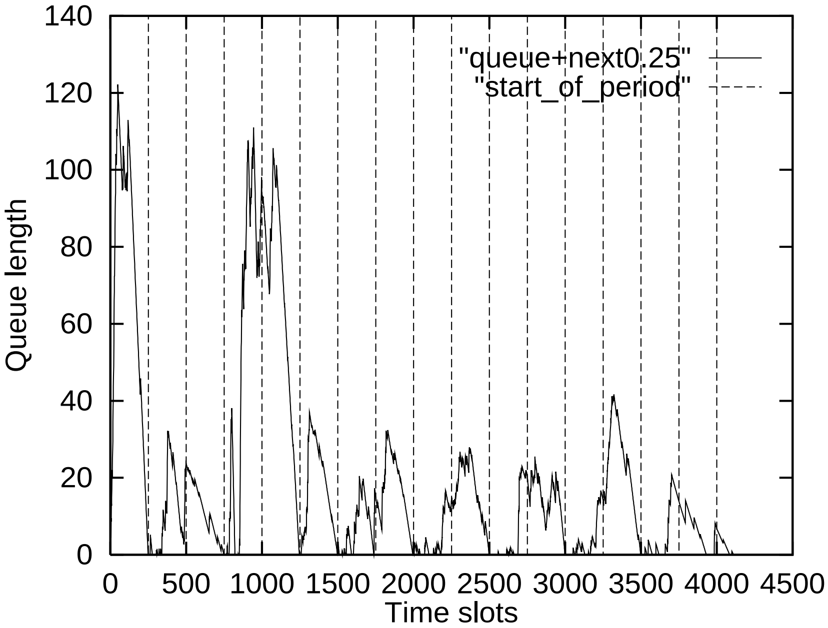


Figure 16: The queue length of `queue+next` for $\rho = 1$ with marks of start of a period.

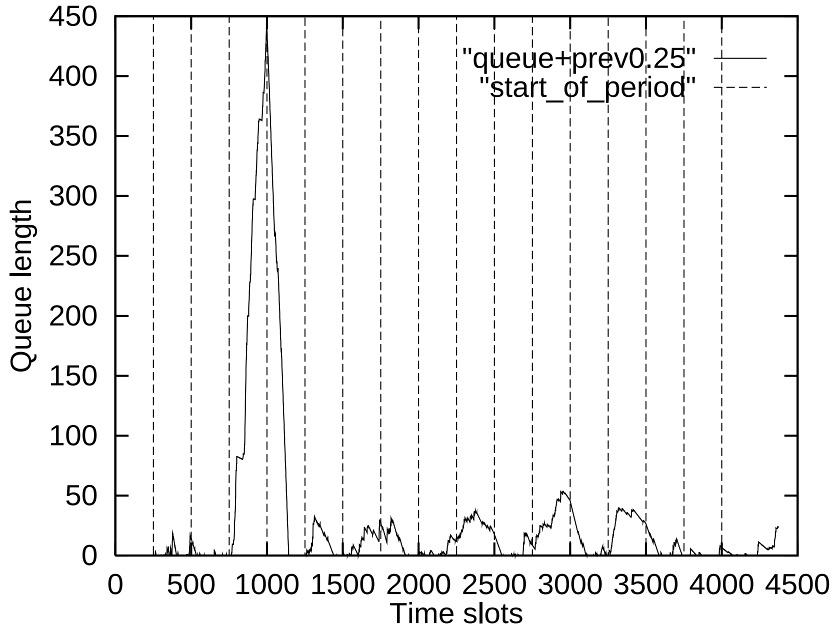


Figure 17: The queue length of `queue+prev` for $\rho = 1$ with marks of start of a period.

Acknowledgments: We would like to thank Edith Cohen and Srinivasan Keshav for introducing the IP-over-ATM problem to us. We would also like to thank Yoav Freund and Steven Phillips for helpful discussions.

References

[SK] H. Saran and S. Keshav, *An empirical evaluation of virtual circuit holding times in IP-over-ATM networks*, Proc. Infocom, 1994.

[LPR] C. Lund, S. Phillips, N. Reingold, *Adaptive Holding Policies for IP over ATM networks*, Submitted to ACM SigComm 1994, February 1994.

[JR] B. B. Juang, L.R. Rabiner, *An Introduction to Hidden Markov Models*, IEEE assp magazine, January 1986

A Theoretic analysis

In order to simplify our analysis we will refer to time in a period both as discrete (and ranges in the set $\{1, \ldots, \tau\}$) and as continuous (and ranges in the interval $[0, \tau]$) according to our needs.

Recall that T_i is the number of packets arriving in period i. Let $T_{max} = max_{i=1}^{n} T_i$ and $T_{avrg} = \frac{1}{n}\sum_{i=1}^{n} T_i$.

Claim 1 *Algorithm* `queue`*, with* $\rho = 1$*, insures that the maximal queue length during a session is bounded by* $2\ T_{max}$*.*

Proof: Consider period i and time slot t, clearly $q_i(t) \leq q_i + T_i$. Since `queue` sends q_i packets in period i, $q_{i+1} = T_i$. This implies that $q_{i+1}(t) \leq T_i + T_{i+1} \leq 2T_{max}$. □

The following claim shows that the bound of Claim 1 is tight.

Claim 2 *There exists a scenario in which the worst case queue length in algorithm* `queue`*, with* $\rho = 1$*, is* $2T_{max}$*.*

Proof: Consider the following session.

$$T_1(t) = \begin{cases} 10 & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases}$$

$$T_2(t) = \begin{cases} 10 & \text{if } t = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$q_1 = 0$$

This gives us $q_2(1) = 20$. □

Next we prove a bound on the average queue length using algorithm `queue`.

Claim 3 *Algorithm* `queue`*, with* $\rho = 1$*, insures that the average queue length in the session is bounded by* $\frac{3}{2}T_{avrg}$*.*

Proof: Note that $q_i(t) \leq q_i + T_i - \frac{t}{\tau}q_i$, since $\mu_i \geq \frac{q_i}{\tau}$. Therefore $\hat{q}_i = \frac{1}{\tau} \cdot \int_0^\tau q_i(t)dt \leq q_i + T_i - \frac{1}{2}q_i = \frac{1}{2}q_i + T_i$. Since $q_i = T_{i-1}$, we have $\hat{q}_i \leq \frac{1}{2}T_{i-1} + T_i$, thus $\hat{q} = \frac{1}{n} \cdot \sum_{i=1}^n \hat{q}_i \leq \frac{3}{2n} \sum_{i=1}^n T_i = \frac{3}{2}T_{avrg}$. □

Claim 4 *There exists a scenario in which the average queue length in algorithm* `queue`*, with* $\rho = 1$*, is* $(\frac{3}{2} - \frac{1}{2n})T_{avrg}$*.*

Proof: Consider the following session:

$$T_k(t) = \begin{cases} 1 & \text{if } t = 1 \\ 0 & \text{otherwise} \end{cases}$$

for k, $1 \leq k \leq n$. This gives $\hat{q}_1 = 1$ and $\hat{q}_i = 1.5$, for $i \geq 2$. Thus the average queue length is $\frac{1}{n}(1 + \frac{3}{2}(n-1))$. □

The next theorem bounds the average queue length of `queue+next`.

Theorem 1 *Algorithm* `queue + next`*, with* $\rho = 1$*, insures that the average queue length in the session is bounded by* $\frac{5}{8}T_{avrg}$

In order to prove the theorem we add the following notations:

$L(S)$ - the duration, in time slots, of the session S.

$T_{[S]}(t)$ - the traffic arriving at time slot t of the session S.

$q_{[S]_i}$ - The queue length at the beginning of period i of session S.

$q_{[S]_i}(t)$ - The queue length at time slot t of period i of session S.

$\hat{q}_{[S]_i}$ - The average queue length in period i of session S.

P_t - the number of the period that contains time slot t, i.e., $P_t = t \text{ div } \tau$.

Y_t - the time slot in P_t corresponding to t, i.e., $Y_t = t \bmod \tau$.

Let S be a session. We note by $t_1 < t_2 < \ldots < t_l$ all the time slots were traffic arrive. Let $t_0 = 0$.

We create $l+1$ sessions, such that session S_i is identical to S up-to time t_i, and after t_i no packets arrive. More formally, for each k, $0 \leq k \leq l$, let S_k be a session such that $L(S_k) = L(S)$,

$$T_{[S_k]}(t) = \begin{cases} T_{[S]}(t) & 1 \leq t \leq t_k \\ 0 & t > t_k \end{cases}$$

We define

$$C_{j,k} = \hat{q}_{[S_k]_j} - \hat{q}_{[S_{k-1}]_j}.$$

In some sense $C_{j,k}$ is the contribution of the packets that arrive in the t_k time slot to the average queue length in period j. The next two lemmas will formalize this notion.

Lemma 1 *For every k and j,* $\hat{q}_{[S_k]_j} = \sum_{i=1}^{k} C_{j,i}$

Proof: The proof, for each j, is by induction on k. For the basis we have $\hat{q}_{[S_0]_j} = 0$, and $\hat{q}_{[S_1]_j} = \hat{q}_{[S_1]_j} - \hat{q}_{[S_0]_j} = C_{j,1}$. For the induction step $\hat{q}_{[S_k]_j} = \hat{q}_{[S_{k-1}]_j} + C_{j,k} = \sum_{i=1}^{k-1} C_{j,i} + C_{j,k} = \sum_{i=1}^{k} C_{j,i}$. □

Corollary 1 *For every j,* $\hat{q}_{[S]_j} = \sum_{k=1}^{l} C_{j,k}$

The next lemma relates the average queue length and the contributions $C_{j,k}$.

Lemma 2 $\hat{q}_{[S]} = \frac{\sum_{j=1}^{n}\sum_{k=1}^{l} C_{j,k}}{n}$.

Proof: By definition, $\hat{q}_{[S]} = \frac{1}{n} \cdot \sum_{j=1}^{n} \hat{q}_{[S]_j}$. By Corollary 1, the lemma follows. $\square$

The following two lemmas show that the contribution $C_{j,k}$ is zero for periods preceding the one of t_k and following two or more periods.

Lemma 3 *If $j < P_{t_k}$ then $C_{j,k} = 0$.*

Proof: The only difference between S_k and S_{k-1} is the packets arriving in the t_k time slot. As $j < P_{t_k}$ there is no difference in the queues length at the j period in both sessions. $\square$

Lemma 4 *If $j \geq P_{t_k} + 2$ then $C_{j,k} = 0$.*

Proof: Consider $j > P_{t_k} + 1$, then for any t, $\forall 1 \leq t \leq \tau$, $q_{[S_k]_j}(t) = q_{[S_{k-1}]_j}(t) = 0$. Thus $\hat{q}_{[S_k]_j} = \hat{q}_{[S_{k-1}]_j} = 0$, and therefore $C_{j,k} = 0$. $\square$

Lemma 5 $C_{P_{t_k},k} + C_{P_{t_k}+1,k} \leq \frac{5}{8} T(t_k)$

Proof: Let $i = P_{t_k}$. Using a similar argument to that in Lemma 3, one can see that $q_{[S_k]_i} = q_{[S_{k-1}]_i}$.

The queue length in S_k are smaller than the ones in S_{k-1} up to time slot t_k because the bandwidth allocated for S_k in period i is greater than the bandwidth allocated for S_{k-1} in period i, the initial queue length q_i is the same in both sessions and till the t_k time slot the arrival manner is identical. Formally, $t < Y_{t_k}$, $q_{[S_k]_i(t)} \leq q_{[S_{k-1}]_i(t)}$.

The difference in bandwidth allocation in the i'th period in both sessions is exactly $\frac{T(t_k)}{\tau}$. Therefore, for each $t, Y_{t_k} \leq t \leq \tau$ we have $q_{[S_k]_i}(t) \leq q_{[S_{k-1}]_i}(t) + (T(t_k) - \frac{T(t_k)}{\tau} \cdot (t - Y_{t_k}))$.

Specifically, at the end of the period, $q_{[S_k]_i}(\tau) \leq q_{[S_{k-1}]_i}(\tau) + (T(t_k) - \frac{T(t_k)}{\tau} \cdot (\tau - Y_{t_k}))$.

In other words, $q_{[S_k]_{i+1}} \leq q_{[S_{k-1}]_{i+1}} + (T(t_k) - \frac{T(t_k)}{\tau} \cdot (\tau - Y_{t_k}))$.

In period $i+1$, the bandwidth allocation of S_k and S_{k-1} is $\frac{q_{[S_k]_{i+1}}}{\tau}$ and $\frac{q_{[S_{k-1}]_{i+1}}}{\tau}$ correspondingly, since $T_{i+1} = 0$ in both S_k and S_{k-1}. Thus, in those periods $q_{[S_k]_{i+1}}(t) = q_{[S_k]_{i+1}} \cdot \frac{\tau - t}{\tau}$ and $q_{[S_{k-1}]_{i+1}}(t) = q_{[S_{k-1}]_{i+1}} \cdot \frac{\tau - t}{\tau}$. Therefore,

$$q_{[S_k]_{i+1}}(t) \leq q_{[S_{k-1}]_{i+1}}(t) + (T(t_k) - \frac{T(t_k)}{\tau}(\tau - Y_{t_k}))\frac{\tau - t}{\tau}$$

Hence, $C_{i,k} + C_{i+1,k} = (\hat{q}_{[S_k]_i} - \hat{q}_{[S_{k-1}]_i}) + (\hat{q}_{[S_k]_{i+1}} - \hat{q}_{[S_{k-1}]_{i+1}})$

$$= \frac{1}{\tau} \int_0^\tau \left(q_{[S_k]_i}(t) - q_{[S_{k-1}]_i}(t) \right) dt + \frac{1}{\tau} \int_0^\tau \left(q_{[S_k]_{i+1}}(t) - q_{[S_{k-1}]_{i+1}}(t) \right) dt$$

$$\leq \frac{1}{\tau} \int_{Y_{t_k}}^\tau \left(T(t_k) - \frac{T(t_k)}{\tau}(t - Y_{t_k}) \right) dt + \frac{1}{\tau} \int_0^\tau \frac{\tau - t}{\tau} (T(t_k) - \frac{T(t_k)}{\tau}(\tau - Y_{t_k}))dt$$

$$= T(t_k) \cdot \frac{1}{2} \cdot (1 + \frac{Y_{t_k}}{\tau} - (\frac{Y_{t_k}}{\tau})^2)$$

Note that $T(t_k)$ and Y_{t_k} do not depend on t, so we have to integrate a linear function. We would like to maximize the above expression. Denote by $x = \frac{Y_{t_k}}{\tau}$, the function $f(x) = 1 + x - x^2$ has a unique maximum at $x = \frac{1}{2}$ where $f(x) = \frac{5}{4}$. Thus $C_{i,k} + C_{i+1,k} \leq \frac{5}{8} \cdot T(t_k)$. □

Proof of Theorem 1: By Lemma 2, $\hat{q}_{[S]} = \frac{\sum_{j=1}^n \sum_{k=1}^l C_{j,k}}{n}$. By Lemmas 3, 4 and 5, we have $\sum_{j=1}^n C_{j,k} \leq \frac{5}{8} \cdot T(t_k)$. Therefore, $\hat{q}_{[S]} \leq \frac{5}{8} \frac{\sum_{k=1}^l T(t_k)}{n} = \frac{5}{8} T_{avrg}$
□

B Another aspect of queue+next vs queue

It seems quite obvious that a periodic algorithm whose aim is to minimize the average queue length should allocate enough bandwidth to empty within the next period the queue it has at the beginning of that period. It also seems quite logical that such an algorithm bandwidth allocation in the i'th period should not be less than $\mu_i = \frac{q_i}{\tau} + \frac{T_i}{\tau}$. In this section we will present some examples showing that some times it is better to allocate $\mu_i = \frac{q_i}{\tau}$ than $\mu_i = \frac{q_i}{\tau} + \frac{T_i}{\tau}$. We will also present examples showing that even this amount of bandwidth, $\frac{q_i}{\tau}$, should not always be allocated.

Consider the following example:
Every odd period 100 packets arrive at the end of the period. Every even period there is no traffic. An algorithm that allocates $\mu_i = \frac{q_i}{\tau} + \frac{T_i}{\tau}$. will have $\hat{\rho} = \frac{1}{2}$ and exactly the same average queue length as an algorithm which allocates only $\mu_i = \frac{q_i}{\tau}$ which in turn has $\hat{\rho} = 1$. If we limit our algorithm so that the utilization is $\hat{\rho} = 1$ or even $\hat{\rho}$ greater than $\frac{1}{2}$ we simply could not allocate $\mu_i = \frac{q_i}{\tau} + \frac{T_i}{\tau}$.

The next example shows that even a bandwidth of $\mu_i = \frac{q_i}{\tau}$ should not always be allocated. In order to simplify our examples, we will consider algorithms bound to work with $\hat{\rho} = 1$. (Similar examples can show that this is also the case for fixed $\hat{\rho}$ different than one). Those algorithm must maintain $\sum_{i=1}^n \mu_i = \frac{1}{\tau} \cdot \sum_{i=1}^n T_i$. We will consider a four periods session.

$$T_1(t) = \begin{cases} 100 & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases}$$

$$T_2(t) \equiv 0$$

$$T_3(t) = \begin{cases} 100 & \text{if } t = \frac{\tau}{3} \\ 0 & \text{otherwise} \end{cases}$$
$$T_4(t) \equiv 0$$

An algorithm which tries to empty the queue it has will allocate $\mu_2 = 100$. It will have to allocate $\mu_3 = 0$, as any other bandwidth it allocates in the third period would be partially wasted, resulting necessarily in $\hat{\rho} < 1$. In the forth period it will necessarily allocate $\mu_4 = 100$. In order to send all packets. Its average queue length will thus be :
$\frac{1}{4} \cdot (50 + 66.66 + 50) = 41.665$

Another algorithm can allocate $\mu_1 = 0$, $\mu_2 = 50$, $\mu_3 = 150$ and $\mu_4 = 0$. This time the average queue length will be:
$\frac{1}{4} \cdot (75 + 8.33 + 33.33) = 29.166$
Which is less than the previous average queue length.