ESPRIT Measure 20113

Deliverable 3.4.2
Performance Evaluation – Controlled Traffic

and

Deliverable 3.4.3
Demonstration and Evaluation

April 1999

Abstract

Implementations of the Measure CAC algorithms on a real local ATM switch are described. A test and demonstration network environment for the algorithms is presented. The algorithems are evaluated with various traffic sources with respect to accuracy and computational complexity. The results show that the Measure CAC algorithms, used on the time scale of end user connections requests are implementable in real time, even with the prototype implementations described here. While the tracking of the arrival process of individual flows, as opposed to simple aggregate analysis, is practical in the experiments we have performed, aggregate analysis has proven to have a higher performance and be more easily implemented. Experiments which compare Measure with other approaches is shown, as are experiments which use Measure with aggregate traffic but use a history mechanism (to forget about performance from the distant past). The history mechanisms used step outside our theoretical framework but may provide a practical way of using aggregates.

Nonetheless, there are questions which remain with respect to our implementation, notably the time scale on which estimations can and should be made. Results for high cell loss rates (1 in 1000) are sastisfactory, but we are not achieving good performance at lower loss rates. Our implementation is not mature and more investigation is required to explore the differences between theoretical and actual behaviour of the algorithms.

1 Introduction

Connection Admission Control (CAC) denotes the set of actions taken by the network during the connection set-up phase in order to accept or reject an ATM connection. A connection request is only accepted when sufficient resources are available to carry the new connection through the network at its requested Quality of Service (QoS) while maintaining the agreed QoS of existing connections. During the connection set-up phase the following information has to be declared, negotiated and agreed between the "user" and "network" to enable CAC to make a reliable connection acceptance/rejection decision:

- A Service Category (such as Constant Bit Rate (CBR) or non-real-time-Variable Bit Rate (nrt-VBR))

- a QoS class expressed in terms of cell transfer delay, delay jitter and cell loss ratio (CLR), and
- specific limits on traffic volume the network is expected to carry.

For a given connection across a network it is not necessary that all these aspects of the CAC decision be declared every time. Many of the parameters, such as (CLR) or cell delay transfer may be implicit for the network on which the connection is being requested. As a result the actual information declared between a prospective "user" and a "network" may be only the service category and a characterisation of the traffic volume limits. An example of this would be that a connection is made on the assumption a certain QoS is available in a network and when it makes its connection it declares that it is a nrt-VBR connection with a specific Peak Cell Rate (PCR), Sustained Cell Rate (SCR) and Maximum Burst Size (MBS).

It can be seen that the algorithm controlling the decision made during the CAC will control the policy of the network. This decision will attempt to balance the requirements of the "user" (achieve the desired QoS) versus the requirements of the "network" (do not violate the QoS guarantees made to other pre-existing connections). As a result of this balance of "trade-offs", a highly pessimistic CAC algorithm may always achieve the QoS by assuming the worst possible characteristics about a new connection. As a result this algorithm would allow few connections into the network; in return for always achieving the QoS commitment, such a CAC algorithm would potentially waste resources – leaving much of the network under utilised. In comparison, a highly optimistic algorithm could always assume the best possible characteristics about a new connection. Such an algorithm would risk violating the QoS contracts made to existing connections for the sake of making maximal use of the available resources. An ideal CAC algorithm will achieve an even balance between "user" and "network".

During the development of such ideal CAC algorithms, substantial effort has been invested in modelling and experimenting with the entire network situation. Models are made of all aspects of the situation including traffic sources, network behaviour, the multiplexing of new connections and the variety of CAC algorithms available. However modelling alone does not satisfactorily assess the behaviour of real CAC algorithms implemented in real situations. Additionally, common modelling techniques involve the use of simulated sources of traffic; these are used because they are well understood and easily generated. However due to the variety of sources and the continual development of new network users (and thus new types of traffic sources), such simulated sources of traffic do not represent adequately the range of behaviours such traffic sources can have in a network.

The inability to model the whole of a real-world CAC process and the inability to adequately represent real sources of traffic mean it becomes desirable to evaluate CAC algorithms in a controlled experimental situation using real traffic sources in an actual network. Through the use of modelled traffic sources a comparison can be conducted between the theoretical models of the CAC situation and the implementation; this enables both the feedback to improve CAC algorithms and a way to ensure realistic assumptions are made in the construction of CAC algorithms. By using real sources of traffic, such as video data streams or client-server file system traffic, the CAC algorithm can be tested against traffic sources that are less easily modelled thereby ensuring their usefulness in the real-world environment.

This document presents the realisation of the *Measure* estimator, describing the performance evaluation of the LAN ATM estimator under controlled and live traffic. We present results evaluating the behaviour of the *Measure* algorithm implemented as a prototype CAC mechanism implemented on an ATM based network. This implementation can be used with both modelled traffic sources and real sources of traffic such as video data. We can obtain traces of ATM cell streams of real traffic which can then be replayed in real time. This allows both realistic traffic to be used and for experiments to be repeatable. The *Measure* implementation has been used to predict the performance of various traffic types and admission decisions sufficiently that a working CAC algorithm has been able to be demonstrated.

Section 2 establishes the terminology and presents the theoretical framework of *Measure*. In Section 3 we describe how the theoretical foundations of Measure can be applied to develop

algorithms for connection admission control. Section 4 details the experimental environment used in the evaluation of the Measure CAC algorithm. The experimental environment has been designed to allow comparison of different CAC algorithms under different traffic loads. The traffic sources used for the experiments of this paper are discussed in Section 5

Section 6 describes specific implementation issues for the *Measure* and *Hornet* algorithms. Section 7 reports results for the experiments we have conducted. These experiments include investigations of the effectives of *Measure* and *Hornet* including a contrast of these algorithms with other CAC mechanisms. Additionally, results that illustrate the behaviour of the *Measure* and *Hornet* algorithms as their parameters are varied are also presented here.

2 Theory

Theoretical Framework

We first establish the terminology that we will use. Our central concern is the loss of cells due to overflow at a buffer. Consider an ATM traffic stream arriving at a buffer which has finite storage capacity b; here b may be sized according to the delay constraint of the traffic. Cells are removed from the buffer at fixed rate s, the *line-rate*. Each traffic stream has a finite duration, as might be expected for calls of finite length. We refer to a traffic stream as a *trace*. Associated with each trace is a *cell-loss ratio* between zero and one; we denote the cell-loss ratio for a buffer-size b and a line-rate s by $\mathrm{CLR}(b, s)$. Experience with a wide variety of traffic-sources shows that the the logarithm of $\mathrm{CLR}(b, s)$ is asymptotically linear in the buffer-size b for practical buffer sizes[1], and for fixed line-rate s. A typical example is shown in Figure 2 which plots, for a set of motion JPEG sources and fixed line-rate s on the Fairisle ATM network at Cambridge [3], the logarithm of the observed cell-loss ratio[2] against buffer-size b.

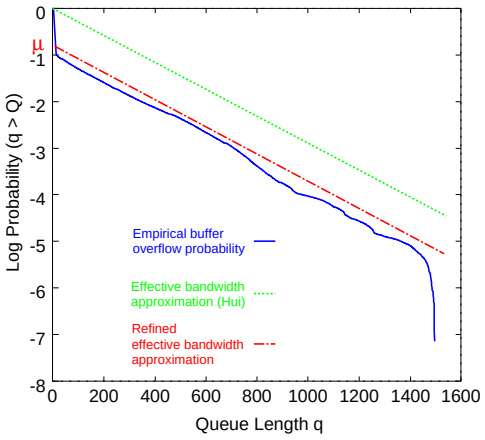


Figure 1: Empirical loss probability for 18 streams of JPEG coded video

We define the *bandwidth requirement* of a trace to be the minimum line-rate at which a target cell-loss ratio c is not exceeded in a buffer of storage capacity b:

$$\mathrm{BWR}(b, c) := \min\{s : \mathrm{CLR}(b, s) \leq c\}. \tag{1}$$

Notice that this is an operational definition which does not involve any statistical theory; for a given trace, it can be determined empirically by trial and error. In practice of course it is neccessary to estimate the bandwidth requirement of a trace rapidly.

[1] We do not intend to fuel the self-similarity debate [1, 2] further – we simply report our observations based on experience with real ATM traffic using practical buffer sizes and time-scales.

[2] Observe that the sharp drop-off at the tail of the distribution is due to the finite observation period of the experiment.

The general features of a plot of the logarithm of cell-loss ratio against buffer-size for fixed line-rate are explained by queuing-theory. The trace is identified with a portion of a sample-path of a stationary stochastic process $\{A_t\}$, the *arrivals process*; here A_t denotes the total number of cells which have arrived up to time t. It can be shown, using large deviation theory, that if the arrivals process has a rate-function $I(a)$ on the scale of t, such that

$$\mathbb{P}(A_t/t > a) \asymp \exp(-tI(a)),$$

then the cell-loss ratio in a buffer of size b with fixed line-rate s decays exponentially in b when b is large:

$$\mathrm{CLR}(b, s) \asymp \exp(-b\delta).$$

Furthermore, the decay-rate δ is determined by the rate-function $I(a)$:

$$\delta = \min\{I(a+s)/a : \ a > 0\}.$$

What we refer to as the *thermodynamic entropy* of a traffic stream is just the rate function $I(a)$. These theoretical results, valid for a wide class of stochastic processes, offer an explanation of the observed asymptotic behaviour of the logarithm of the cell-loss ratio as a function of buffer-size and relate the slope of its linear asymptote to the rate-function of a stochastic process representing the traffic.

The behaviour of the cell-loss ratio for small buffer sizes is less well understood, but theoretical studies of model stochastic processes have thrown some light on this. A simple two-state Markov model is capable of capturing the general features of ATM traffic in a crude way. For such models, Buffet and Duffield [4] showed that the large deviation estimate $\mathbb{P}(Q > q) \asymp \exp(-q\delta)$ can be improved by the introduction of a prefactor $e^{-\varphi}$ so that we have a bound valid for all values of q:

$$\mathbb{P}(Q > q) \leq \exp(-\varphi - q\delta).$$

This has motivated us to use a straight-line upper bound to the graph of the logarithm of the cell-loss ratio against buffer-size:

$$\log \mathrm{CLR}(b, s) \leq -\mu(s) - \delta(s)b, \tag{2}$$

at least for buffer sizes above a small threshold b_0. The parameters $\mu(s)$ and $\delta(s)$ are functions of the line-rate s. Suppose we have estimates $\hat{\mu}(s)$ and $\hat{\delta}(s)$ of the parameters; we define the *estimated bandwidth requirement* by

$$\widehat{\mathrm{BWR}}(b, c) := \min\{s : -\hat{\mu}(s) - \hat{\delta}(s)b \leq \log c\}. \tag{3}$$

This is used in two ways in our CAC algorithms: for the existing multiplexed traffic, we use on-line estimation of the parameters μ and δ to determine its estimated bandwidth requirement; for a proposed new connection, we set μ equal to zero and estimate δ using declared parameters.

The on-line estimation of the slope parameter δ for multiplexed traffic is the characteristic feature of the CAC algorithms introduced in this document. The on-line estimation of the intercept parameter μ is comparatively straight-forward, and based on the frequency of occurrence of small interarrival times; we describe it in more detail in [5]. The estimation of δ is based on the formula

$$\delta(s) = \min\{I(a+s)/a : a \geq 0\},$$

relating it to the large deviation rate-function $I(a)$ of the arrivals process. This formula has an equivalent version which is more suited to estimation:

$$\delta(s) = \max\{\theta > 0 : \lambda(\theta) < s\theta\},$$

where $\lambda(\theta)$ is the *scaled cumulant generating function* (SCGF) of the arrivals process, defined by

$$\lambda(\theta) = \lim_{t\to\infty} (1/t) \log \mathbb{E}(\exp(\theta A_t)). \tag{4}$$

It is of crucial importance to observe that the SCGF is automatically convex.

A Note on Effective Bandwidth. Before describing the CAC algorithms, we briefly clarify our use of the term effective bandwidth, or equivalent capacity. The definition which most closely fits our approach originates with Hui [6], and is the value of s in equation (2), which describes a straight line of slope δ through the origin in the graph of the empirical CLR of Figure 2. This gives rise directly to the *effective bandwidth approximation*, which is the function

$$\hat{s}(b,c) = -b\frac{\lambda(-\log c/b)}{log(c)}. \tag{5}$$

The effective bandwidth approximation is an upper bound on the rate at which a buffer of size b must be drained in order to meet the target CLR constraint c. It does not include any refinement involving the intercept μ, shown in Figure 2. We have termed our refined approximation the estimated bandwidth requirement, as defined in equation (3). However, we shall make use of the effective bandwidth approximation of equation (5) in situations where we have no way to estimate μ, for example, when presented with a new call attempt for which we have no measurements.

3 The Measure CAC Algorithms

In this section we show how the theoretical foundations of Measure can be applied to develop algorithms for connection admission control. The algorithms work by estimating scaled cumulant generating functions (SCGFs) of arrival processes. The relevant arrival processes are those of the traffic which is currently being carried (which Measure estimates) and of the traffic for which admission is requested.

Parameters supplied by a new connection request are passed to the predictor which attempts to predict the SCGF, and thence the bandwidth requirement, of the new connection. This is combined with the estimated bandwidth requirement of the current multiplex, produced by the *estimator*, to decide whether or not to admit the connection.

Figure 2 shows the behaviour of the *Measure* CAC algorithm. Given a current multiplex of connections, the system estimates the bandwidth requirement of the multiplex from the estimated SCGF using the entropy estimator (depicted here by a thermometer). The predictor used in the *Measure* algorithm requires a new connection attempt to declare only its peak rate P; this is used as a pessimistic assessment of the connection's bandwidth requirement. The CAC algorithm sums the peak rate P, and the estimated bandwidth requirement of the multiplex; if the total is less than the link capacity, then the connection can be accepted without violating the QoS of any connections. As soon as the new connection commences, the estimator uses measurements of the new multiplex to revise its estimate of the current bandwidth requirement. When the next connection attempt arrives, the procedure is repeated, as shown. If a new connection attempt arrives before the algorithm has developed an accurate estimate of the new bandwidth requirement, as shown in Figure 3, the algorithm acts *conservatively*. It uses the most recent stable estimate of the bandwidth requirement, plus the sum of the peak rates of all subsequently admitted connections. Thus, in Figure 3, the second connection will be rejected, because the sum of the two peaks $P + P'$ and the first bandwidth requirement estimate exceeds the link capacity.

3.1 Measure Algorithm

The *Measure* algorithm uses the following formulation as the estimator for the SCGF λ of the current multiplex:

$$\lambda(\theta) := \lim_{t\to\infty} \frac{1}{t} \log \mathbb{E}(e^{\theta A_t}), \tag{6}$$

An obvious way of estimating the SCGF from observations of A_t over a finite time interval T is to use empirical averages to estimate the expectation: we break the observations into K blocks

of length B in time and let $\tilde{X}_k$ be the total arrivals in the k^{th} block. The SCGF is then readily estimated as follows:

$$\hat{\lambda}(\theta) := \frac{1}{B} \log \frac{1}{K} \sum_{k=1}^{K} e^{\theta \tilde{X}_k}. \tag{7}$$

3.2 Hornet

The *Hornet* algorithm differs from the *Measure* algorithm only in that it allows more information about a new connection to be supplied. (In fact *Hornet* can be seen as a superset of *Measure*.) The over conservative behaviour of *Measure* can be moderated by using more information about the connection.

Arriving traffic is often described by crude parameters, possibly just the peak rate, or possibly by the ITU (and ATM Forum) defined Generic Cell Rate Algorithm (GCRA). Traffic conforming to GCRA(T,τ), if passed through a queue of size τ/T served at at rate $1/T$, will not cause overflow. Traffic may be forced to conform to several GCRA constraints. Note that GCRA constraints appear in both ITU and ATM Forum standards for traffic control in ATM networks, and that policing a source to ensure that it obeys a set of GCRA constraints is simple and is currently performed in many switches.

The CAC algorithm works as follows. At all times an estimate of the effective bandwidth of the streams passing through a queueing point in the network is available. Let the difference between the total capacity and the estimate, that is the available capacity, be c. Let the total buffer available be b. Then a bound may be produced on the required bandwidth of the incoming stream and compared with c. Several possibilities then arise:

1. if only the peak rate (sometimes referred to as peak cell rate, PCR) of the new stream is available, then set the required bandwidth estimate to the PCR:

   Accept the connection if $c \geq$ PCR; otherwise,

   Reject the connection.

2. if a single GCRA constraint, GCRA(T,τ) is given, then set the effective bandwidth to

   $$1/T \text{ if } b \geq \tau/T,$$

   otherwise set the effective bandwidth to the line rate at the source.

   Accept the connection if $c \geq 1/T$ and $b \geq \tau/T$; otherwise,

   Accept the connection if $c \geq$ source line rate; otherwise,

   Reject the connection.

3. when the ATM Forum parameters PCR, SCR and IBT (peak cell rate, sustained cell rate and inter burst tolerance) are given, the traffic conforms to the GCRA constraints GCRA(T,τ) and GCRA(T',0), with $T' = 1/\text{PCR}$, $T = 1/\text{SCR}$, and $\tau = \text{IBT}$. We can assume that $T > T'$.

   If the buffer is greater than τ/T then the effective bandwidth is the SCR. Otherwise the effective bandwidth is very nearly the PCR. More precisely:

   Accept the connection if $b \geq \tau/T$ and $c \geq 1/T$; otherwise,

   Accept the connection if $c \geq (\tau - bT + bT')/\tau T'$; otherwise,

   Reject the connection.

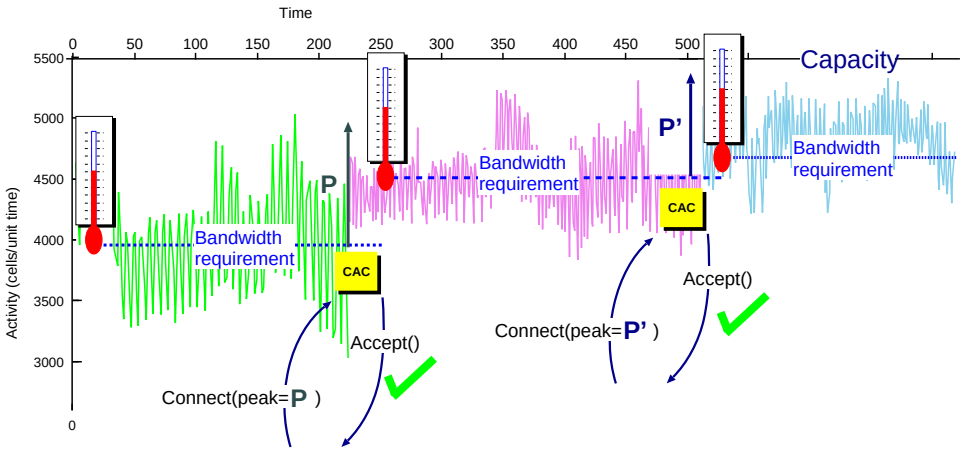


Figure 2: Operation of the Measure CAC algorithm

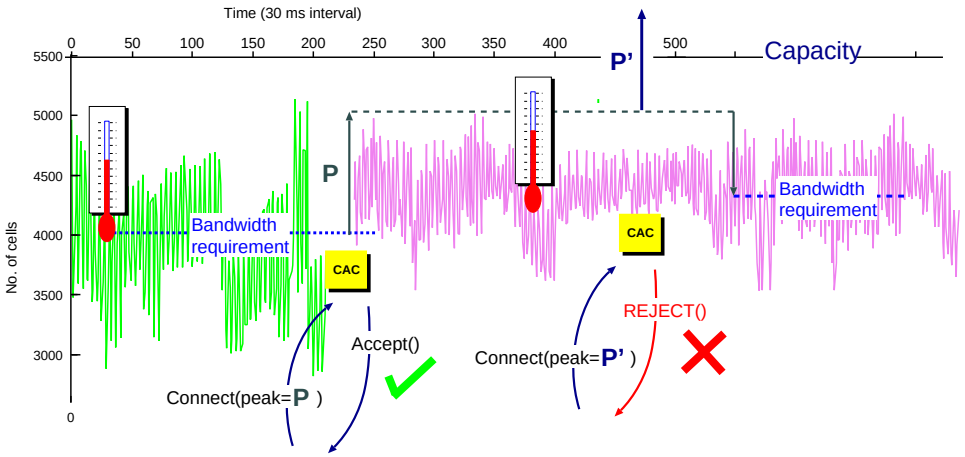


Figure 3: Conservative CAC using the Measure CAC algorithm

4 Evaluation & Demonstration Environment

The experimental environment used in the evaluation of the Measure CAC algorithm is described in this section. Figure 4 shows the implementation architecture adopted for the CAC algorithm(s) which the Measure project has developed. The experimental architecture is such that it enables us to implement and test not only the specific CAC algorithms described in this paper, but also any other algorithms developed in the Measure project, and algorithms which have been proposed by other authors. We aim, using this testbench, to evaluate and compare a range of CAC algorithms from the literature, to test how well the Measure algorithms perform in practice.

At the heart of the architecture is an ATM switch, which is instrumented to provide measurements of the form suitable to the Measure and other measurement based CAC algorithms.

Input traffic into the switches used for the study comes from a series of custom built traffic generators which are able to generate traffic comprising the statistical multiplex of well over a hundred ATM traffic sources, each of which draws its traffic behaviour either from an analytical model (such as the two-state Markov model used widely in our earlier simulation studies) or a trace of activity, such as the transmission of MPEG and Motion JPEG streams from ATM Cameras, and the other traces measured in Deliverable 3.1. These traces include NFS, Web, X windows and other traffic, and have already been used with our estimators. The generators are based on PCs running the Nemesis Operating system, part of the ongoing development of which is being sponsored under Workpackage 4 of this project. Each traffic generator is capable of saturating the ATM transmission links should this be required, and the whole suite of generators gives us unrivalled flexibility in experimentation.

The instrumentation on the ATM switch enables us to retrieve measurements of a quality suitable for use in the Measure algorithms. A Measurement controller, running off the network on a unix workstation, is responsible for collecting the data from the switch and storing it appropriately. The measurement controller measures not only the traffic activity, which is the input to the measure algorithms, but also the QoS experienced by the traffic: its CLR, queue length distributions, inter-cell loss times, and other measures which allow us to compare the performance of the system in practice with the observed perfromance of a simulation of the system using a simulated switch model, offline.

The CAC system works as follows: a call generator (running on unix) is responsible for 'generating' according to some distribution, or perhaps a trace of measured arrivals, the arrival of new calls. New calls may be of multiple types, and each call may randomly (according to some distribution) determine its call type and any set of parameters which it is required to present. In the case of Mosquito, the calls need only declare their peak rate, however for full ATM Forum compliant calls, we might require a call to declare its SCR, IBT and any other useful parameters. New calls generated by the traffic generator arrive at the CAC decision system when they are generated. Each call presents its parameters to the CAC system and requests a connection to be set up across the switch. The CAC system, in turn, uses measurements from the switch of the current activity, together with the currently loaded CAC algorithm (Measure, Mosquito, Peak rate allocation, Kelly/Key etc) to make a decision as to whether or not to admit the call. Only one CAC algorithm may be in operation at any time.

If a call is admitted, the CAC algorithm will reply to the call generator accepting the call. The call generator then instructs the trafic generator controller to 'set up' a new traffic generator with the appropriate parameters for a call fo this type. The call type might be on-off, or some other analytical model, or trace driven. The traffic generator controller then starts the new call by instructing it to begin transmission. The generator then adds the cell stream from the new call to the cell stream from the full set of ATM connections for which it is responsible, and simply transmits its total load. Each connection has a lifetime which is drawn from some distribution or perhaps a set of measurements, and when its lifetime expires, the traffic generator responsible for it stops its transmission of cells, and the call 'clears down'.

It is important to stress that in this setup there is no real ATM signalling. The processes running off-switch assume the full load of the 'signalling' and therefore we can emulate the arrival

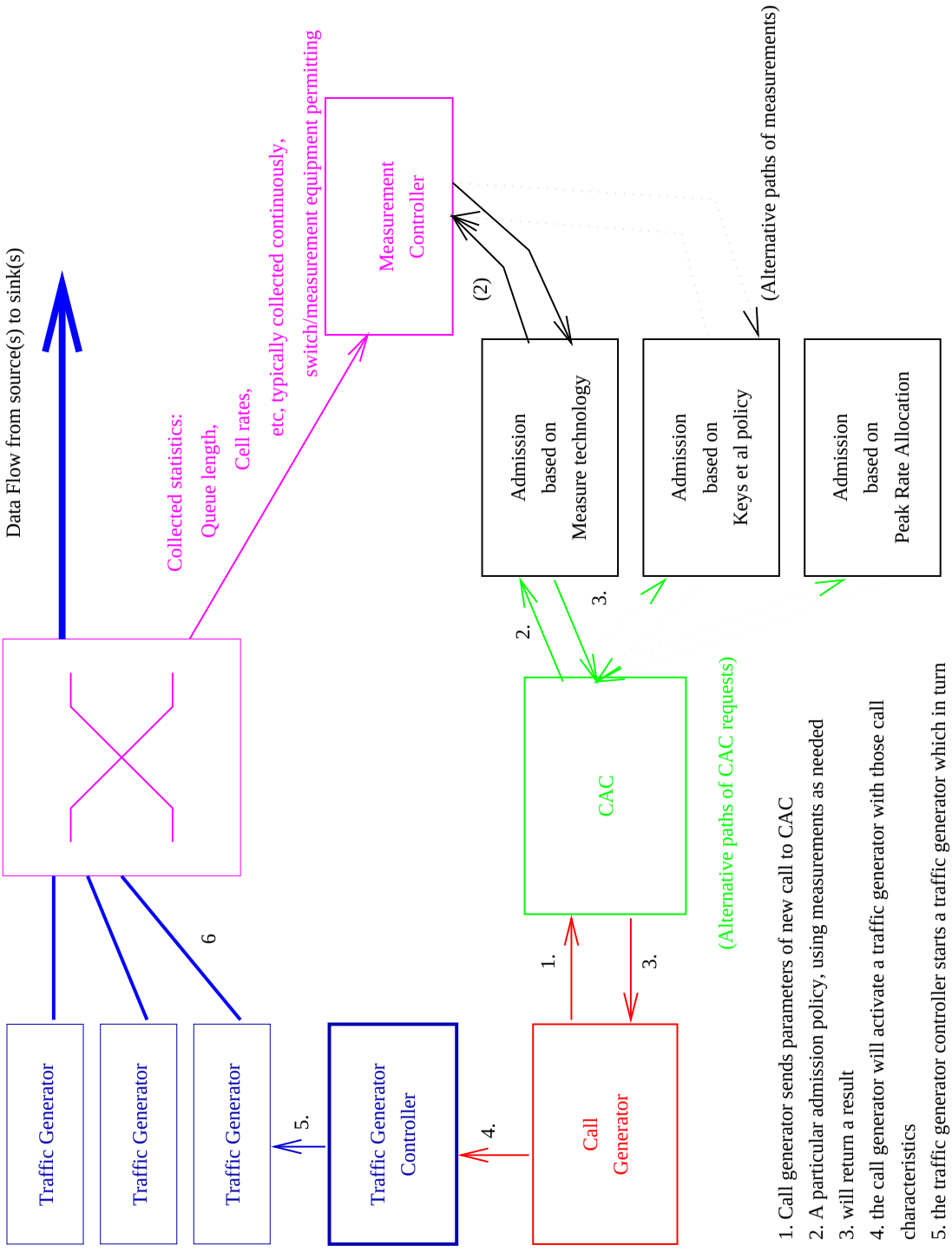


Figure 4: Architecture of the CAC implementation

of calls at rates far higher than could be sustained by any real ATM signalling implementation. Also, the setting up of a 'connection' through the switch is optimised in the sense that all VCI/VPI pairs to be used in an experiment are mapped through the switch before the experiment starts, meaning that we do not need to actually set up paths during the experiment. This greatly aids efficiency, and meas that we can achieve realistic call setup rates for large networks.

Our experimental system is equipped with a facility which provides graphical output in real time, showing several critical performance parameters for the system under study as calls arrive and depart from the network.

Figure 5 shows two screendumps of the system, depicting the evolution of an experiment over time. The graphical output system has 3 graphs, stacked vertically. The x axis in all cases is time, shown in seconds since the start of the experiment. This grows as the experiment continues. Most experiments last for in excess of an hour (ie we run the system for long enough to obtain accurate statistical measures of the system perfomance under each traffic type, switch parameters and CAC algorithm). Details of the results from this system will be presented in a later deliverable, as our efforts to date have concentrated on developing the first implementation.

The top graph in each set of 3 produced by the on-line system shows the call arrival process. For each call which arrives a vertical bar is drawn. In the event that an arriving call was accepted by the CAC algorithm, a green vertical bar is drawn. If the call is rejected, then the vertical bar is red, and extends downwards. In the leftmost series of graphs, no calls have been rejected because this shows the system in a startup transient, before there is sufficient traffic in the system for calls to be rejected. In the right-hand set of graphs, however, calls begin to be rejected after 100 seconds.

The second graph from the top in each set is a display of the traffic dynamics in the switch, measured in real time. Each graph shows 3 lines. The green line is the measured instantaneous utilisation of the link being studied. The orange line is the output of the on-line estimation algorithm of the effective bandwidth (the Bandwidth Requirement) of the traffic, as estimated by the online estimation algorithm in use. In this case the algorithm is the simple Measure algorithm described earlier. Typically the BWR line is below the instantaneous demands, as would be expected. Finally, the purple line depicts the 'committed bandwidth': the sum of the estimated BWR as described above, plus the amount of resource commited, for example by using the peak rate, for all calls admitted since the most recent stable estimate from the estimation algorithm. Thus the purple line is a conservative estimate of the resource requirements of the total traffic multiplex given the active sources and the new calls which have been accepted but for which no measurements have been made yet.

Finally, the bottom graph in each set shows the number of calls in progress in the system, over time. This rapidly climbs, as new calls enter the system at a greater rate than they clear down, and because in the empty system (at left) no calls are rejected. Once rejections occur, the number of calls in progress stabilises, but displays the expected variation due to statistical fluctuations. In this experiment the calls are 2-state on-off sources, with a peak rate of 10 Mb/s, a mean rate of 1 Mb/s, and a mean burst length of 25 cells. In the experiment shown, the Measure algorithm is being used, with a CLR constraint of $10^{-}3$ for the entire multiplex. The CLR constraint is never violated, yet the link utilisation is close to optimal for this traffic.

It is important to note that an observational error is introduced as the measured CLR value is reduced. A measured CLR of 1×10^{-3} means that in an experiment of 100,000,000 cells, 100,000 cell loss events will be counted. However a measured CLR of 1×10^{-6}, for the same length experiment, represents only 1,000 cell loss events. To give the same number of events for a smaller CLR value, the length of the experiments would need to be increased by an appropriate amount. An experiment that transmits 100,000,000 cells requires about 2 hours to run to completion; to achieve the same number of loss events for a measured CLR of 1×10^{-3}, the experiment would need to transmit 100,000,000,000 cells and run for 2,000 hours (about 12 weeks). Obviously this is not realistic and as a result many more experiments have been run at lower values of CLR, with the majority targeting a CLR of 1×10^{-3} or 100,000 cell loss events in 2 hours.

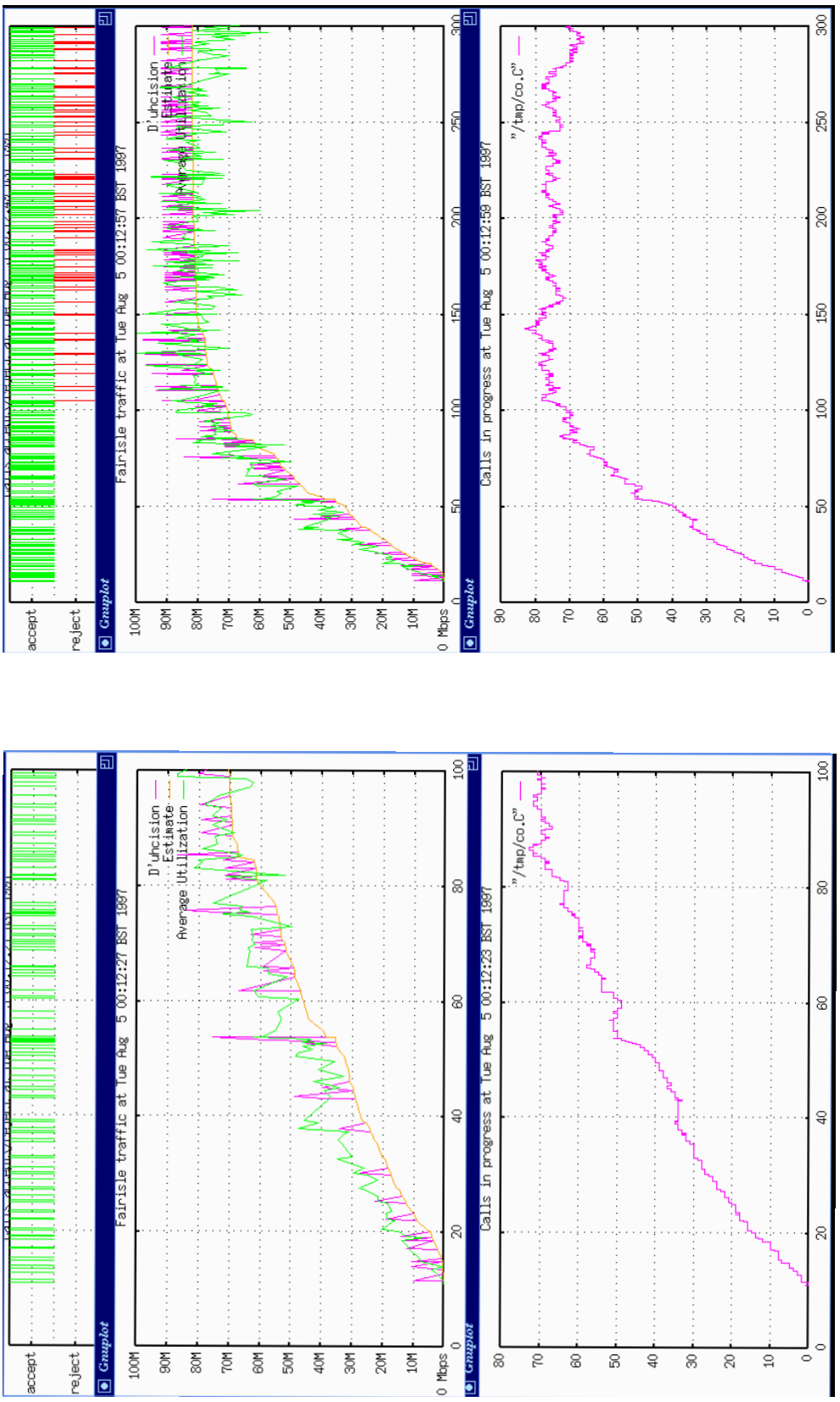


Figure 5: Screen shots of the CAC system in on-line operation

5 Traffic

In this section we detail the traffic sources used for the experiments of this paper.

5.1 TP10S1

TP10S1 is a traffic source based upon a two-state ON-OFF Markovian model. The theoretical model used is shown in Figure 6. The burst sizes and inter-burst spacings each have an exponential distribution with means derived from the MBS and SCR respectively. In the on-state, the cells of a burst are emitted at PCR.

The behaviour of traffic from the source is based around the uniformly distributed random variable X and the traffic properties of PCR, SCR and MBS. The variable X in turn, is based on a pseudo-random number generator. As a result, several traffic generators can have consistent traffic properties of PCR, SCR and MBS yet, through the use of different seeds to the pseudo-random number generator, the generators will create a stream of cells that will differ over time in the cell level characteristics of burst length, burst size and inter-burst spacing.

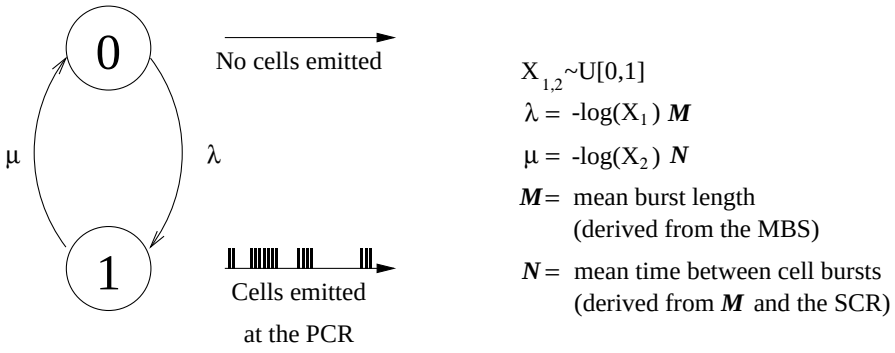


Figure 6: Markov 2-state on-off generator.

The TP10S1 traffic type has a PCR of 10Mbps and an SCR of 1Mbps. The cell bursts themselves have an MBS of 25 cells. This means cell bursts will be transmitted at 10Mbps, the exponentially distributed cell bursts have a mean length of 25 cells and the exponentially distributed inter-burst spacing gives an SCR of 1Mbps.

5.2 StarWars2

StarWars is a source which has been widely used in performance studies of ATM systems in the literature, namely a trace of the activity of the Star Wars movie. The Star Wars data set was produced by Garrett and Vetterli at Bellcore, and has been studied in some detail, for example in [1, 9] and [10]. It comprises the information content in bytes per frame for about 2 hours of the film, as transmitted by a DCT based codec similar to JPEG. The byte count per frame is broken down into "slices" of 16 video lines. With 30 slices per frame and 24 frames per second, one slice represents the information transmitted in about 1.4 milliseconds. The Star Wars traffic has been shown to exhibit signs of long-range dependent behaviour, making it potentially difficult for a measurement based CAC to cope with.

The Star Wars traces were constructed by encoding each slice as a single AAL5 PDU which was transmitted (and traced); cells from each slice were transmitted at a CBR rate equal to the slice rate, reducing the peak rate from the line rate to about 24.2 Mb/s for the worst slice. The mean rate is about 5.3 Mb/s.

5.3 VP10S4

VP10S4 is a traffic stream based, like the StarWars stream, upon a video source. For VP10S4, the video used was a combination of programs on local television, including news broadcasts, talk shows and feature movies. The peak rate is about 10 Mb/s and the mean rate is about 4.0 Mb/s.

5.4 T2MIX

T2MIX is two-state Markovian source of the same style as TP10S1. It emitts cells at a constant (peak) rate of about 11.1 Mb/s (28.8Kcells/s) in the ON state, and is silent in the OFF state.

There is a range of parameters for this traffic source with each new connection choosing its parameters randomly: the mean (as a fraction of the peak) is chosen from 0.1, 0.3, 0.5, 0.7, and 0.9 with probabilities 1/16, 4/16, 6/16, 4/16, and 1/16 respectively. The burstiness is chosen from 0.3, 0.5, 0.7, and 0.9 with probabilities 1/8, 3/8, 3/8, and 1/8 respectively.

6 Algorithm Implementation

While both the *Measure* and *Hornet* algorithms have elegant mathematical descriptions, the implementation of these systems is complex. This complexity derives from there being many different choices the implementor may follow in the course of constructing working versions of these algorithms. This section describes these choices in the implementation of the *Measure* and *Hornet* algorithms described previously in Section 3.

Section 6.1 details the rational for breaking up the implementation into two components, a front-end and a back-end. There are two front-end implementations one each for *Measure* and *Hornet* and there are two back-end implementations one each for per-connection measurements and aggregate measurements. The front-end and back-end delineation falls neatly onto the algorithms boundary between predictor and estimator. Section 6.2 covers details of the *Hornet* front-end implementation. Section 6.3 discusses the back-end implementation based upon per-connection line measurements, while Section 6.4 compares that with a back-end implementation based upon aggregate line measurements.

Section 6.4.1 addresses issues raised through the use of an aggregate line measurement implementation specifically and comments on the unbounded nature of *Measure* based algorithms in general. Finally, Section 6.5 gives an assessment of the complexity of implementation.

6.1 *Measure*

The *Measure* algorithm uses continuous measurements of the line utilisation to refine its estimate of the bandwidth required by the multiplex of traffic on a network.

Reiterating the conclusions of Section 3, the SCGF, and thus for a given δ value, the effective bandwidth is estimated from observations of A_t over a finite time interval T using empirical averages to estimate the expectation. We break the observations into K blocks of length B in time and let $\tilde{X}_k$ be the total arrivals in the k^{th} block. The SCGF is estimated as follows:

$$\hat{\lambda}(\theta) := \frac{1}{B} \log \frac{1}{K} \sum_{k=1}^{K} e^{\theta \tilde{X}_k} \tag{8}$$

In the simplest implementation of the algorithm, measurements of the activity of the line are collected and for each CAC decision, the SCGF is re-calculated incorporating the new measurements.

However, in a practical implementation this would involve two drawbacks. Firstly, the buffering of an unknown amount of data prior to each recalculation would involve an unknown and unbounded amount of memory. Secondly, the calculation of the SCGF itself may involve an unknown and unbounded amount of time to perform. This would introduce an unknown and variable delay in the response of the CAC proces. Rather than an approach that required the calculation of

the effective bandwidth estimate for each and every admission attempt at the time of the attempt, an alternative method was required.

Rather than performing the calculation of the effective bandwidth estimation for each CAC decision, an alternative method is to divide the task of collecting measurements and calculating the new effective bandwidth estimation from the task of performing the CAC decision. Figure 7 illustrates this technique where the effective bandwidth estimate is calculated independently to the CAC decision. This division of labour of the CAC algorithm into a front and back end means that the CAC decision process becomes independent, asynchronous, to the calculation of the effective bandwidth estimate.

The front and back ends still need to share information between them and this is done through the sharing of a block of memory. It is into this memory that the back-end places the latest copy of the bandwidth estimate.

By breaking the algorithm into two distinct components, we can bound the amount of CPU time and memory that either component will require. In particular this means that the CAC decision process can be performed asynchronously of the effective estimate calculation; thus the period between consecutive CAC decisions is not the minimum time it takes to calculate an effective bandwidth estimate.

The technique of breaking up of an algorithm into two parts in this way is a suitable approach for any algorithm that may rely upon synchronous access to measurements. The *Measure* algorithm relies upon synchronous access to measurements in order to supply the most up to date estimate of the effective bandwidth; other algorithms similarly may require this access to estimate information.

Table 6.1 presents a code snippet illustrating the admission process for a new connection in *Measure*. The effective_bandwidth variable used in calculating the current_estimate is calculated by the back-end component of the algorithm. Note that *Measure* only requires that incoming connections declare their maximum possible utilisation, PCR.

6.2 *Hornet*

In the calculation of the effective bandwidth estimate, *Hornet* and *Measure* are identical; in this way both algorithms share their back-end components. The difference between *Hornet* and *Measure* comes in the implementation of the decision process in the front-end. While *Measure* can only process the simplest traffic description: the peak transmission rate of that source – PCR; *Hornet* is able to use the ATM Forum parameters PCR, SCR and IBT. *Hornet* does not need all these parameters to be declared by a new connection but if a new connection does declare these parameters *Hornet* can make a better attempt at admitting the new connection.

By operating on the additional parameters offered by a new connection attempt, *Hornet* has the potential to operate the system closer to the maximum line rate, thereby achieving the best utilisation of available network resources. *Hornet* does this while still keeping the multiplex of connections within the desired target CLR. Table 6.2 presents a code snippet illustrating the admission process for a new connection in *Hornet*.

6.3 Per-connection measurements

The actual mathematics for the calculation of the effective bandwidth estimate from the set of traffic samples, $\tilde{X}_k$,are the same for both the Per-connection measurement back-end and the Aggregate measurement back-end. The difference between these implementations is that the Per-connection implementation is more strictly in accordance with the expectations of the algorithms' architects. The Per-connection implementation ensures that every block, $\tilde{X}_k$, only contains traffic for connections that were active over the time covered by that particular block. In contrast, and as discussed in the next Section, aggregate measurements may contain measurements of traffic for connections that have long since been 'torn-down'.

For the Per-connection implementation, we must ensure each block, $\tilde{X}_k$, is based upon only active connections, this means that when a connection is completed and 'torn-down', the line

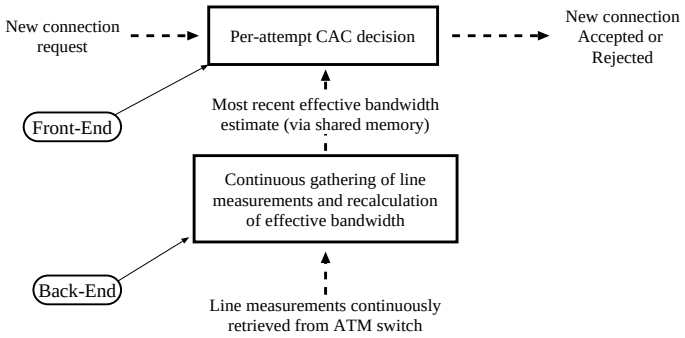


Figure 7: Concurrent recalculation of effective bandwidth estimate and CAC decisions using a pair of processes: front-end and back-end.

```
 newConnectionAttempt( newCallPeakRate )
...
   current_estimate = /* current effective_bandwidth (from back-end)
                         + the PCR of all connections that have
                         entered the system since the current
                         effective_bandwidth was calculated */

    if((line_capacity - current_estimate) < newCallPeakRate) {
        /* Connection attempt has been rejected */


    }else{
        /* Connection attempt has been accepted */

        /* record the PCR of the new connection along
           with the time it entered the system */

        /* allocate connectionID */

    }
...
```

Table 1: Code snippet illustrating the admission process for a new connection in *Measure*.

```
newConnectionAttempt( newCallPeakRate [,tau ] [,newCallMeanRate] )
...
    /* calculate GCRA parameters */

    T = 1 / newCallMeanRate;

    T_prime = 1 / newCallPeakRate;

    current_estimate = /* current effective_bandwidth (from back-end)
                          + the required bandwidth of all connections
                          that have entered the system since the current
                          effective_bandwidth was calculated */

    if((line_capacity - current_estimate) < newCallPeakRate) {

        /* 'Measure' style admission failed */

        if(current_estimate + newCallMeanRate > st->maxlinerate ||
           buffer_length < (tau / newCallMeanRate) ) {

            /* 1st round 'Hornet' failed */

            /* now calculate (tau - (buffer_length x 1/newCallMeanRate) +
            (buffer_length x 1/newCallPeakRate))/(tau x 1/newCallPeakRate) */


            effective_rate =
                (( tau - (buffer_length * T) +
                   (T_prime * buffer_length))
                 / (tau * T_prime));

            if( line_capacity - estimate >=  effective_rate ) {

                /* 2nd round 'Hornet' failed
                   new connection attempt has failed */

            }else{
                /* new connection attempt succeeded */
            }

        }else{
            /* new connection attempt succeeded */
        }

    }else{

        /* new connection attempt succeeded */
    }

    if( /* new connection attempt succeeded */ ) {

        /* record the required bandwidth of the new connection along
           with the time it entered the system */

        /* allocate connectionID */
    }

...
```

Table 2: Code snippet illustrating the admission process for a new connection in *Hornet*.

utilisation statistics that it was responsible for are removed from all the blocks before the effective bandwidth is recalculated. The Per-connection measurement implementation has significant overheads associated with it because we need to keep information about which connections are active for each block as well as the original measurements for each connection over each block. This is necessary because the block must either be recalculated without including the completed connections' statistics or have the completed connections' statistics removed from the sum.

Per-connection measurements, as the name implies, requires that statistics about the contribution to the line of each individual connection is available from the switch. Without a mechanism to identify what data belongs to which connection, the per-connection mechanism cannot identify and remove the appropriate data. The per-connection measurements back-end can be used with either the *Measure* or *Hornet* front-ends.

6.4 Aggregate measurements

The incentive for an implementation based upon aggregate rather than per-connection measurements was the massive potential load that moving measurements between the switch statistics system and the component calculating the *Measure* estimates is contained. The amount of data transferred for the lower range of values of the size of the block length B and thus the granularity of measurements (the period over which measurements are made) could make aggregate measurements the only plausible implementation approach to take. For each port carrying 256 connections and a block length B of 100 cell times in length, measurements made every 100 cell times will require over 2Mbyte/second be moved into the *Measure* system for each port of the switch. If only aggregate measurements were used this figure would be reduced to 2Kbytes/second for the same measurement period. It was with this information in mind that an aggregate measurement implementation was considered worthy of investigation.

In direct contrast to the per-connection measurements back-end of Section 6.3, the aggregate measurements back-end requires a measurement of the total line activity for the given block length period. This means the back-end, and indeed the measurement gathering system are both significantly simplified. This significant simplification can be seen clearly in the comparison of the code of per-connection measurement code Table 6.3 with the aggregate measurement code of Table 6.4

The drawback of the aggregate measurement based back-end is that each block, $\tilde{X}_k$, contains measurements that have been caused by connections that may have since been 'torn-down'. In this way the aggregate measurements may contain old data no longer relevant to the current traffic mix and thus not relevant to the calculation of a current effective bandwidth estimation. Such an approach does not have a good theoretical foundation, however the ease of implementation that this approach offers makes it worth investigating. The workload on the switch is also minimised making this mechanism very appealing; supplying line activity statistics for every active connection with the regularity required by *Measure* may simply not be realistic for a large switch with tens or hundreds of thousands of connections across hundreds of ports. However, offering high resolution aggregate measurements might be more realistic for both new switch systems to be designed to offer and old switch systems to be re-engineered to make newly available.

An immediate drawback of aggregate measurements is that it is not clear when the measurement is no longer relevant. In per-connection measurements, a block is no longer relevant when it contains no data for current connections; when this is the case it should be discarded being of no further use in the calculations to predict the behaviour of the current connections. The use of aggregate measurements requires a way to decide when blocks should no longer be included in the set of samples used to calculate the effective bandwidth estimate.

The use of history boundaries to set the oldest point beyond which blocks would be discarded is the subject of the Section 6.4.1.

```
...

   for /* each block period */ {

       for /* each valid connectionID */ {

           perBlock_traffic_sample +=
                perconnection_traffic_sample[ connectionID ];

       }
   }

    /*
       Using all perBlock_traffic_sample with
       at least one valid connection present;
       calculate current effective_bandwidth estimate.

       place this estimate in shared memory to allow
       access by the front-end

       place a time-stamp of when this latest value
       became available in shared memory to allow
       access by the front-end */
...
```

Table 3: Code snippet illustrating the Per-connection measurements based *Measure* back-end.

```
...

     /* perBlock_traffic_samples = an aggregate of
        all line activity at the time the block was taken */

     /*
       Using up to the maximumOldestBLock perBlock_traffic_sample;
       calculate current effective_bandwidth estimate.

       place this estimate in shared memory to allow
       access by the front-end

       place a time-stamp of when this latest value
       became available in shared memory to allow
       access by the front-end */
...
```

Table 4: Code snippet illustrating the Aggregate measurements based *Measure* back-end.

6.4.1 History boundaries

The use of aggregate measurements in general invokes a problem that the content of each block, $\tilde{X}_k$, may contain measurement data for connections that have since been removed from the network. Additionally, if a block does not contain measurement data for any active connections its contribution to the *Measure* calculation of effective bandwidth estimate will be unpredictable at best and unhelpful at worst. In the per-connection implementation, only data for active connections is used to calculate the value of each block, $\tilde{X}_k$; even when this means recalculating the values of each $\tilde{X}_k$ whenever a connection is removed from the system. For aggregate measurements we have two problems, firstly, that some blocks will contain measurements that are a mixture of connections that are still active and connections that are no longer active, and secondly, that some blocks will contain measurements for no current data at all.

A solution to both of these problems is to place an upper limit on the lifetime of any particular block: once a block becomes older than a certain period it is removed from the traffic mix. In this way if this history length is shorter than the average length of a connection, some blocks with valid data will be discarded, whereas if the history length is longer than the average length of a connection, some blocks will be included in the *Measure* calculation that contain no valid data at that time. From this we can see that selecting the history length may in itself be a non-trivial task.

In addition to using a fixed history Grossglauser & Tse [11] explored using an adjustable and dynamic history length based upon the number of connections in progress and the mean length of each connection. While their work was not with the *Measure* algorithm, it was widely considered directly relevant to any algorithm that has at its basis the sampling of connection activity over the lifetime of connections. The boundary on the block history in time is given as:

$$T_{max} = \frac{m}{\sqrt{n}} \tag{9}$$

The (trivial) implementation of this code is given in Table 6.4.1.

This value for the boundary on the length of history implies that as more connections enter the system, historical knowledge of longer connections is discarded in favour of blocks containing traffic mixtures for more recent connections. This history mechanism is used for all experiments based upon the aggregate measurement back-end unless otherwise mentioned.

The additional use of a fixed upper boundary on the total number of blocks may also be needed for real-world implementations of both the per-connection and aggregate measurement back-end. This is because without such a boundary, a connection held open permanently may cause the total block history to be unbounded. By being unbounded, both the calculation of the *Measure* estimate and the memory required to store the block history would be unbounded. The current implementations of the per-connection or aggregate measurement back-end do not implement such a feature.

6.5 Implementation complexity

In the wider acceptance of the *Measure* algorithm one problem may be its perceived complexity. While increased familiarity with the algorithm foundations and assumptions makes implementation less difficult, such a complex algorithm may have a significant code investment based associated

```
...

      maximumOldestBlock = (meanCallHoldingTime) /
                           sqrt(numberOfCallsInProgress);
...
```

Table 5: Code snippet to calculate the oldest block to be added to an Aggregate measurement.

with it. Table 6 gives an assessment of the complexity of the code. These results are based on the size of the code base and a subjective assessment of the difficulty making the implementation.

From Table 6 we would have to conclude that even the most complicated algorithm based upon *Measure* or *Hornet* could be considered relatively easy. In the context of switch software which can run to many hundreds of thousands of lines, the code investment in these algorithms is not trivial but certainly straightforward.

Algorithm	Lines of Code	Difficulty to Implement 1 (easiest) – 5 (hardest)
Measure Aggregate Back-end	1500	4
Measure Per-connection Back-end	2050	5
Hornet Per-connection Front-end	250	3
Measure Per-connection Front-end	200	2
Peak Rate Allocation	170	1
Gibbens/Kelly95 - III	280	4

Table 6: Implementation complexity given by the amount of code in each CAC algorithm and a subjective assessment of the difficulty of making the implementation.

7 Results

In this section we present results for a range of experiments we have conducted. We have conducted experiments investigating the effectives of *Measure* and *Hornet*, contrasting these algorithms with other CAC mechanisms. Additionally, results that illustrate the behaviour of the *Measure* and *Hornet* algorithms as their parameters are varied are also presented here.

Section 7.1 presents results showing the performance of the implementations in particular contrasts of the desired CLR with the measured CLR. Section 7.2 reports on the behaviour of algorithms when input parameters to the algorithm, in particular the block length B, are varied over a range of values. Section 7.3 presents performance statistics for the components of the *Measure* and *Hornet* CAC implementations as well as performance statistics of comparison CAC algotihms. Finally, in Section 7.4 results comparing the two *Measure* and two *Hornet* CAC algorithms along with *Gibbens/Kelly95 - III* a CAC from [12] and a simple *Peak Rate Allocation* policy.

Unless it is specifically stated, all experiments conducted had 100 cell buffer and a target CLR of 1×10^{-3}. The connection attempts arrive at a high (Poisson) rate with a mean of 10 connection attempts per second. Blocked connections are lost, but the high arrival rate means that the system is continually faced with new connection attempts. Connections have an exponentially distributed length; connections have a mean length of 10 seconds.

7.1 Algorithm Effectiveness

An attraction of a CAC constructed upon the *Measure* or *Hornet* algorithm is that we should be able to set the value of δ: namely the desired CLR and the system buffer size, and the CAC algorithm should attempt to achieve that CLR as a mean for all connections sharing the line. This section presents results showing the performance of the implementations in particular contrasting the desired CLR with the measured CLR.

As discussed in Section 2, both the *Measure* and *Hornet* algorithms are pessimistic about the behaviour of newly admitted connections. Each algorithm assumes the newly admitted connection will perform with the worst possible traffic characteristics that have been declared; as a result the measured CLR of the line is expected to be lower than the desired CLR parameter. In the case of *Measure*, new connections need only declare their PCR, as a result *Measure* is expected to return a higher divergence in the CLR values than *Hornet*: where incoming connections can declare a greater number of parameters.

For the following experiments a buffer size of 100 cells and a target CLR of 1×10^{-3} is used unless otherwise noted. For the results of Table 7 and Table 8, the other adjustment of the Measure algorithm, the Block length B is set to 5×10^{-3} seconds; this parameter is discussed in greater detail in Section 7.2.

Using the ON-OFF traffic source TP10S1, described in Section 5.1; the results of Table 7 were gathered. Connection attempts arrive at a high (Poisson) rate with a mean 10 attempts s^{-1}. Blocked attempts are lost, but the high arrival rate means that the system is continually faced with new connection attempts. As a result it is expected that the system will remain at close to maximum utilisation. Connections have an exponentially distributed length with a mean duration of 10 s.

Each CAC will attempt to achieve as high a line utilisation as possible, however in return for greater line utilisation, greater cell-loss will present itself. Because of this the results of Table 7 and Table 8 need to be assessed only as an attempt by the *Measure* or *Hornet* algorithms to achieve a given CLR; this alone is not the only assessment criteria of a CAC algorithm and a comparison of other aspects of each experiment is given in Section 7.4.

From Table 7 we can see that, with the exception of the *Hornet* algorithm using aggregate measurements, each algorithm returns quite acceptable CLR values measured from the test environment. The *Hornet* algorithm using per-connection line measurements admitted a larger number of connections and thus the resulting measured CLR is greater. The *Hornet* algorithm based upon aggregate line measurements returns a much lower CLR value, almost an order of magnitude lower, indicating that the effect of aggregation of measurements is causing the algorithm to be more pessimistic in its admissions.

Algorithm	Achieved CLR			
	Mean	Var	Std. Dev.	95 % CI
Measure Aggregate	7.075×10^{-4}	2.78×10^{-8}	1.667×10^{-4}	1.414×10^{-5}
Measure Per-connection	9.312×10^{-4}	1.511×10^{-8}	1.229×10^{-4}	6.017×10^{-6}
Hornet Aggregate	2.052×10^{-4}	2.193×10^{-9}	4.683×10^{-5}	3.765×10^{-6}
Hornet Per-connection	5.628×10^{-3}	7.991×10^{-7}	8.939×10^{-4}	8.053×10^{-5}

Table 7: A comparison of desired CLR and achieved CLR for the *Measure* based algorithms using the TP10S1 traffic source.

Using connections carrying the video stream traffic source VP10S4 (described in Section 5.3) with the same connection attempt characteristics as the previous experiment, the results of Table 8 were gathered. The *Hornet* and *Measure* algorithms based upon aggregated line measurements both have performed quite well, while the two algorithms based upon per-connection line measurements have faired less well. The reasons for this difference have turned out to be several factors including: the value of the block length B, the traffic and even the selection of the target CLR have each played a role in the behaviour exhibited.

Algorithm	Achieved CLR			
	Mean	Var	Std. Dev.	95 % CI
Measure Aggregate	1.043×10^{-3}	6.551×10^{-8}	2.560×10^{-4}	2.168×10^{-5}
Measure Per-connection	7.247×10^{-3}	1.618×10^{-6}	1.272×10^{-3}	1.180×10^{-4}
Hornet Aggregate	1.252×10^{-3}	3.094×10^{-7}	5.563×10^{-4}	4.712×10^{-5}
Hornet Per-connection	1.672×10^{-2}	4.496×10^{-6}	2.120×10^{-3}	1.971×10^{-4}

Table 8: A comparison of desired CLR and achieved CLR for the *Measure* based algorithms using the VP10S4 traffic source.

Firstly we chose to test if the algorithms all performed in the same way across a range of target CLR values. Figure 8 graphs the results for experiments where the value of the target

CLR is varied across a wide range of values. The block length B was 2×10^{-3} seconds for these experiments. From this graph it is clear that, for this selection of traffic, block length B and connection load attempt, the performance is poor. However, it is interesting to note that there is a clear relationship between measured and target CLR for each algorithm.

Figure 9 presents the results for a similar comparison of measured and target CLR but using a different traffic source. In this case the video based source VP10S4 is carried by the new connections. Figure 9 presents a situation where the difference between the measured and target CLR values are worse than in Figure 8.

One difference between the VP10S4 and TP10S1 sources is the level of activity (the ratio of PCR and SCR values). One avenue of investigation was that the source itself had a serious affect on the behaviour of the algorithm. It has already been discussed in previous deliverables that traffic with significant Hurstiness properties had been postulated to interfere with the algorithms operation. Subsequently our first choice was to compare high and low activity traffic from an ON-OFF source with better Hurstiness characteristics.

Figure 10(a) and Figure 10(b) each show that in terms of measured CLR, the performance of the *Hornet* Aggregate and *Hornet* Per-connection algorithms improve when the activity of the traffic source is reduced. While it appears the Hurstiness of the traffic of VP10S4 may have been the contributing factor, the block length B used in these experiments is still much larger than those used in any theoretical work. As a result the selection of the block length is the primary contributing factor to the CLR performance reported in Figure 9 and Table 8.

Some of the causes for the differences in performance in some of these experiments is discussed further in Sections 7.2 and 7.3. While the traffic characteristics make a significant contribution; the selection of the block length B also requires considerable thought. Because of the importance of the block length B used by the algorithm the role it plays in the algorithm was investigated at length and the next section presents results that demonstrate the effects arising from changing this variable.

7.2 Algorithm Parameters

In this section we discuss the results of experiments where input parameters to the algorithm, in particular the block length B, are varied over a range of values. The block length B is selected to be large enough to contain a representative sample of traffic; each sample of block length B being statistically independent. However, the block length B cannot be so large that the sample will smooth out short term transients. It is obvious from the outset that the selection of the block length B could vary between traffic types as well as between mixes of traffic, additionally the length of connections as well as their arrival rate also will have an effect on block length B.

From the results of the previous section we have seen that the block length B value may have a significant effect on the behaviour of the *Measure* and *Hornet* CAC algorithms. Figure 11(a) graphs the relationship between the measured CLR and the block length for several of the CAC algorithms using the ON-OFF traffic source TP10S1. Figure 11(b) illustrates the same results using the video based source VP10S4. When we first viewed this data, the 'maxima' feature of the *Measure* algorithms drew our attention. In particular, at first it was not clear why the measured CLR should decline when the block length B was greater than a certain value. The answer was in the design of the *Measure* algorithm; the situation also exists in the *Hornet* algorithm.

The *Measure* algorithm will assume that an incoming connection is transmitting cells at the PCR value declared when the connection was attempted. The algorithm will continue to assume the new connection is transmitting cells at the PCR rate until the next effective bandwidth estimate is available. The next effective bandwidth estimate will not be available until the current block of data (of period B) has been completed. Prior to the availability of a new estimate, the *Measure* algorithm will add to the currently available bandwidth estimate the PCR of the newly connections. This PCR component is removed once the new effective bandwidth estimate is available; since the new connection has data that forms part of the most recent block period. If the block period is significantly slower than the rate at which connections are attempted; the CAC algorithm will turn away a significant number of these new connection attempts.

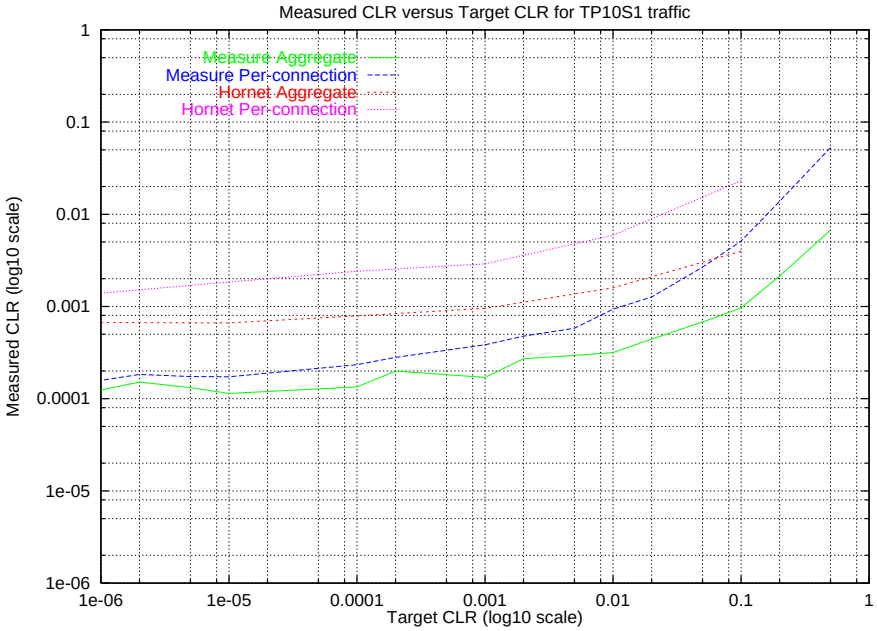


Figure 8: Measured CLR versus target CLR using traffic type TP10S1.

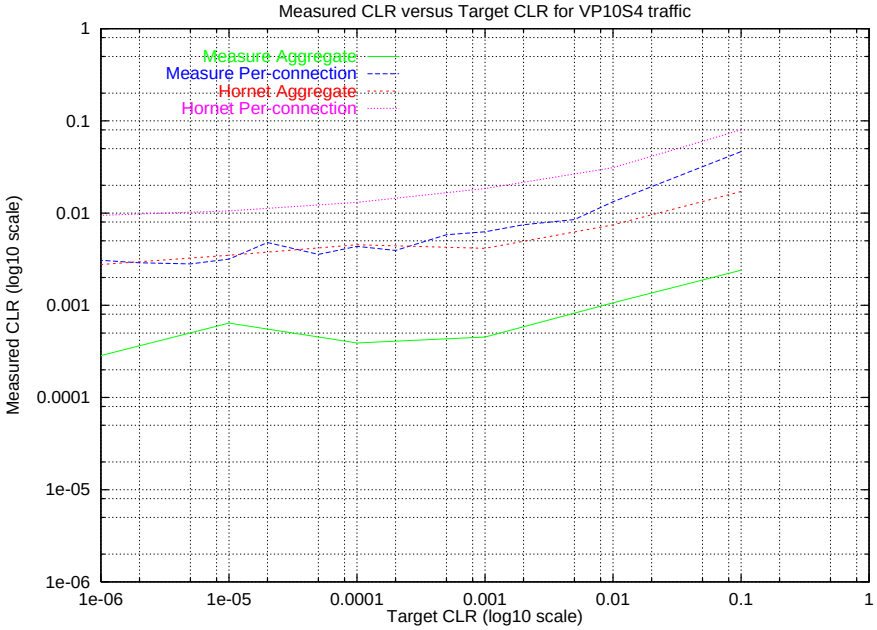
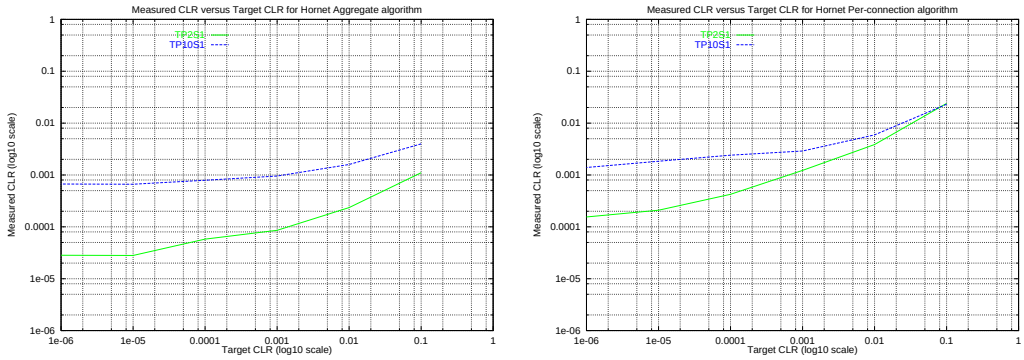


Figure 9: Measured CLR versus target CLR using traffic type VP10S4.



(a) *Hornet* Aggregate CAC algorithm

(b) *Hornet* Per-connection CAC algorithm

Figure 10: Measured CLR versus target CLR for sources with different activity levels.

This situation can be seen in Figure 12, the screen-dump of two running systems: one with a small value for block length B and the other with a large value for block length B. The layout of the screen-dump is described in detail in Section 4. In this figure we can see that for the long block there are steps in the CAC decision line; these steps occur with the same period as that of the block length B itself.

For comparison Figure 12 also contains a screen-dump of an experiment with a short block length and it is obvious that the behaviour of the algorithm is completely different, the number of connections in progress is much higher and, we can assume, the CLR of the system would also be higher.

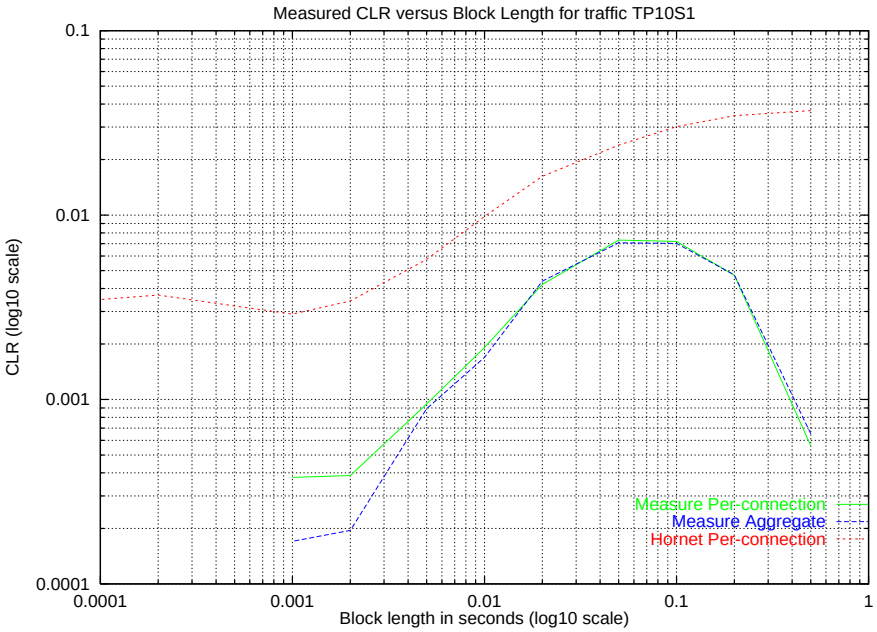
The situations where the *Measure* algorithm performs fewer admissions as the block length B increases can neatly account for the 'maxima' seen in Figure 11(a). The *Hornet* algorithm will also exhibit this characteristic but the effect is reduced because *Hornet* will be able to admit connection attempts that have declared parameters in addition to the PCR used by *Measure*. The result is that a 'maxima' may exist but it will not be as pronounced, this is what we see in Figure 11(a).

What is seen here is that because the *Hornet* and particularly the *Measure* algorithms each have a pessimistic assumption, PCR based admission in the case of *Measure*, when connections first arrive; they will do pessimistic admission for the period of the block length B. The result is as the block length B increases the algorithm becomes increasingly pessimistic. In the most extreme case, an overly long block length B will cause the algorithm to behave as if it were one based solely upon Peak Rate allocation of the line resources.

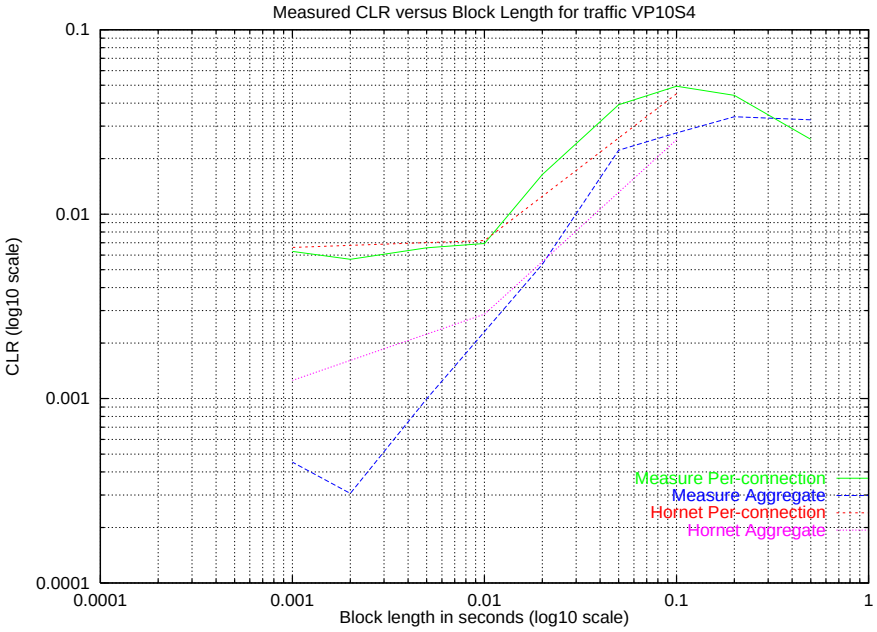
Figure 11(a) and Figure 11(b) also show that as the block length B diminishes, the measured CLR approaches the target CLR. It has always been understood that the selection of the block length B plays a significant role in the behaviour of the CAC algorithm. We can easily obtain results for large values of block length, however obtaining results for small values of block length presents considerable difficulty.

In obtaining smaller and smaller block length B values, implementations are limited by their method as well as the mechanics of the measurement extraction, recalculation of the effective bandwidth estimate and so on, this issue is discussed further in Section 7.3. However, as was seen in Figure 12, the block length B can adversely effect the number of connections the *Measure* algorithm will admit.

A comparison of Figure 11(a) and Figure 11(b) also reveals that the selection of the block length B was a substantial contributor in the difference in results between the results of Table 7 and Table 8 of Section 7.1. It was not expected that the properties of the traffic would have such a significant effect on the algorithm, and provided a suitable value of block length B is selected this appears to hold true. For the two traffic sources VP10S4 and TP10S1 it becomes apparent that different block length B values were required.



(a) traffic type TP10S1



(b) Traffic type VP10S4

Figure 11: Measured CLR versus Block length

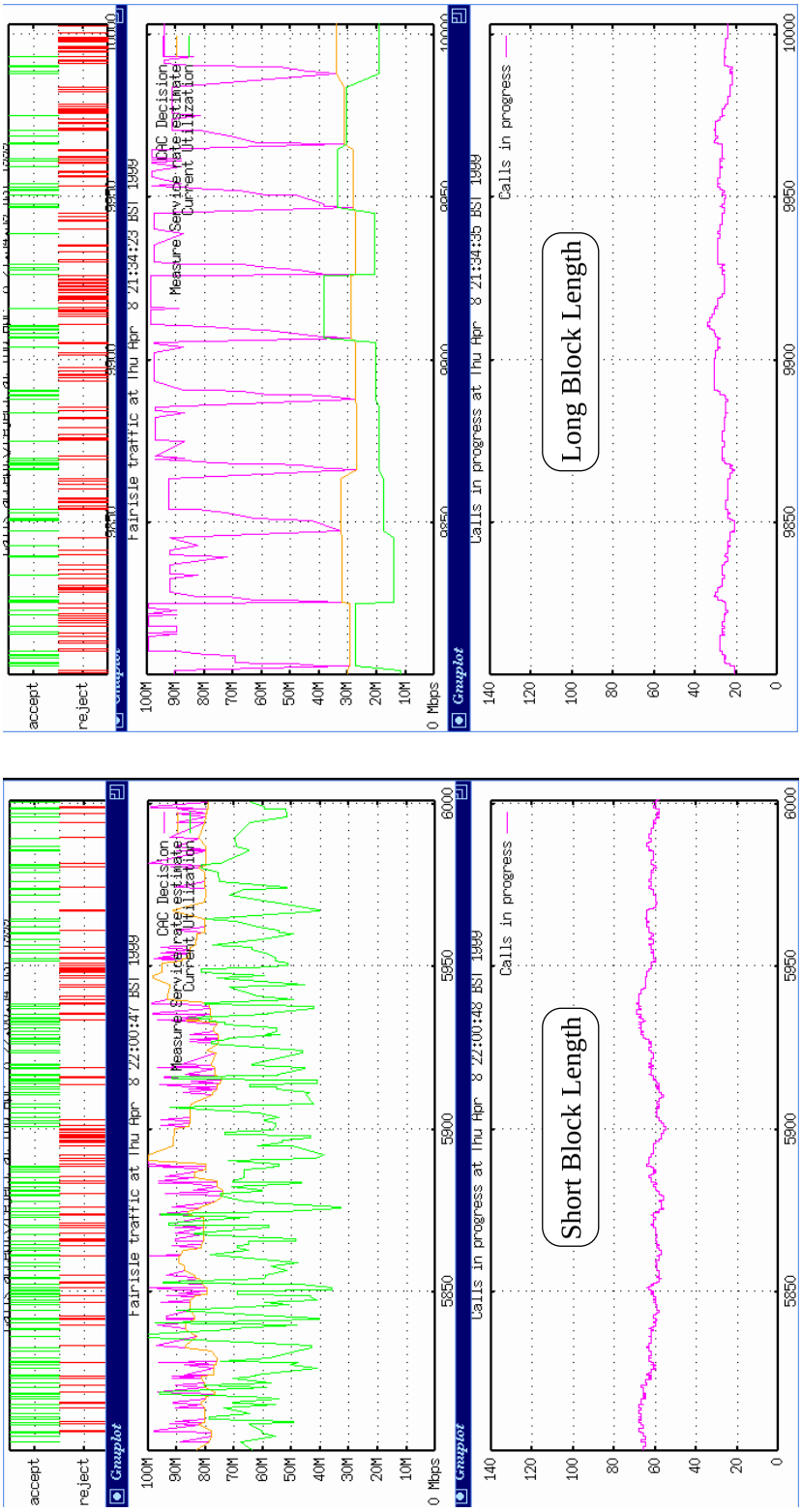


Figure 12: The effect of a short or long block length.

The results illustrated in Figure 13 show the measured CLR for a range of block length values. In this experiment we compare the same traffic type under two different connection attempt loads. In comparison with the previous experiments where the mean rate of connection attempts is 10 per second, the mean rate is set to 0.5s. In order to maintain the same high traffic load on the system the Mean CHT of these connections is 200 seconds. The objective of this experiment is to show that the measured CLR will more closely match the target CLR of the experiments as the connection attempts arrive less frequently.

Clearly Figure 13 shows that the the form of each curve is similar, but that in the case where the load is lower, the CLR results for all values of block length have been reduced. The conclusion we draw from this is that with fewer connection attempts being made, the likely-hood of the most recent estimation of effective bandwidth, being available is increased, as a result the CAC algorithm decisions give an improved measured CLR.

Such a conclusion seems at odds with the previous conclusion that larger values of block length B will cause the algorithm to be more pessimistic and thus result in a **lower** measured CLR. However Figure 11(a) shows a 'maxima' of measured CLR values and thus the measured CLR will increase for increasing block length B values, while the value is on the left of the 'maxima' (or less than about 60ms). Once the block length B is greater than 60ms these results indicate that the pessimistic component of the algorithm becomes the primary issue in the measure CLR value. Reducing the block length B to the desired value is not currently an option for us either; implementation limitations mean a physical lower limit on the block length B value.

While it may be possible to run simulations with small block length B values; there is a distinct lower limit on this value for a physical implementation. We have found that in our physical implementation we are unable to extract per-connection or aggregate measurements from the switch for periods smaller than 1ms. The result is that this places a very real limit on how small the block length B can be. In simulations, values approaching 50-150 cell times have been used, but at between 220μs and 660μs such values are not able to be replicated in this implementation.

The lower limit on the granularity of measurements (the period over which measurements are made) and thus the lower limit on the size of the block length B is often imposed by the switch system itself or by the mechanism which extracts the measurement from the switch. The input of many, up to 256 connections for a typical switch port, of line statistics for one experiment alone means transferring blocks of data 1024 bytes in size into the *Measure* system. If the block length B were a value approaching 100 cell times in length; this would imply that over 2Mbyte/second was being moved into the *Measure* system, and this is only for a single port of the switch. From this example we can see there are engineering problems that will confront implementers who wish to take full advantage of this algorithm.

From this we conclude that while not possible in the current implementation, it would be worthwhile using a smaller value of block length B. We conclude that the current lower limit on block length B may, for some traffic types, mean that the traffic sample taken in that period is too long, smoothing out short term transients that are of importance as input to the *Measure* (and *Hornet*) algorithms. In Section 7.3 we discuss how the mechanics of an implementation will also effect the algorithm behaviour.

7.3 Algorithm Mechanics

The block length B may not be the biggest obstacle to improved performance in the *Measure* algorithm. In addition to collecting measurements over the block length B period, the algorithm must speedily calculate a new effective bandwidth estimate. As we noted in Section 6.1, the collection of measurements and the calculation of the effective bandwidth estimate is a process that occurs continuously in the implementations' back-end. The back-end collects the measurements and once it has collected a block length B period of masurements, a recalculation occurs. This recalculation will take a finite amount of time and this Section details how the behaviour of the recalculation can have an important effect on the effective bandwidth and thus the CAC algorithm.

In this Section we present typical performance statistics for the components of the *Measure* and *Hornet* CAC implementations as well as performance statistics of comparison CAC algotihms.

We show that poor performance in the implementation can lead to poor behaviour of the CAC algorithm.

Prior to a new effective bandwidth estimate being available, a block length B period of measurements needs to be made, and a calculation of the new effective bandwidth estimate so as to incorporate those measurements. We note that any time taken to perform the recalculation is added to the block length B before a new estimate can be used. While it may be able to parallelise these two components, performing a recalculation of the estimate while making the next block length B worth of measurements, if the recalculation takes significantly longer than the block length B period, the estimate will be available less frequently than every block length B period.

A slow recalculation of the estimate will mean the estimate will not be available as often. We note that until the CAC algorithm has a new estimate available it assumes that the newly admitted connections are still at PCR. It is not until the new estimate is made available, incorporating traffic from the newly admitted connection, that the PCR of these recent new connections is no longer part of the decision for new connections. In the worst-case, an overly long recalculation period will cause the algorithm to behave as if it were solely based upon Peak Rate allocation of the line resources. These symptoms are near-identical to the situation that occurs when the block length B is too long.

Table 9 presents the time taken for the calculation of each new estimate, in the case of back-ends and the time taken for each new admission, in the case of front-ends. For comparison the time taken for each new admission is also given for a simple Peak Rate Allocation algorithm and CAC method based on algorithm III from [12]. The *Gibbens/Kelly95 - III* algorithm cannot be given a target CLR, rather its response is tuned with a scaling value, the behaviour of this value must be uncovered by experiment.

In the case of the back-end periods and the *Gibbens/Kelly95 - III* algorithm periods, this time includes obtaining measurements. It is worth reiterating that the *Peak Rate Allocation* algorithm requires no measurements, working solely with the declared PCR of the incoming connection. The *Gibbens/Kelly95 - III* algorithm requires a single aggregate measurement of line activity for each admission process. The *Measure* back-ends require either Per-connection measurements or Aggregate measurements as indicated.

The *Hornet* and *Measure* algorithms share common back-ends subsequently results are only given for the two types of *Measure* back-ends. For both the *Measure* and *Hornet* algorithms their Per-connection and Aggregate front-ends are identical.

Algorithm	Mean	Var	Std. Dev.
Measure Aggregate Back-end (identical for *Hornet* Aggregate Back-end)	2.995×10^{-2}	8.641×10^{-4}	2.939×10^{-2}
Measure Per-connection Back-end (identical for *Hornet* Per-connection Back-end)	1.748×10^{-1}	1.672×10^{-2}	1.293×10^{-1}
Hornet Per-connection Front-end (identical for *Hornet* Aggregate Front-end)	5.748×10^{-5}	1.510×10^{-9}	3.886×10^{-5}
Measure Per-connection Front-end (identical for *Measure* Aggregate Front-end)	1.882×10^{-5}	5.183×10^{-6}	2.277×10^{-3}
Peak Rate Allocation	1.350×10^{-5}	4.231×10^{-11}	6.504×10^{-6}
Gibbens/Kelly95 - III	4.950×10^{-5}	3.856×10^{-9}	6.209×10^{-5}

Table 9: Time taken for each phase of each CAC algorithm (seconds).

To have implemented *Measure* Per-connection as an algorithm that calculated a new estimate as part of each admission process, the total amount of time required would have, on average wait of 174.8ms would have been incurred for every connection attempt. These figures appear to justify the design decision detailed in Section 6, to split the *Measure* and *Hornet* CAC algorithms into front and back-ends.

Figure 8 shows that values of block length B greater than 60ms will cause the CAC algorithm

to become increasingly pessimistic. As concluded earlier in this Section, if new estimates are not made faster than every 60ms then the algorithm also becomes increasingly pessimistic. Table 9 indicates that the *Measure* Per-connection Back-end is, on average, calculating new estimates of the effective bandwidth estimate every 174.8ms, substantially slower than the 60ms of Figure 8.

Figure 14 graphs the measured CLR for a range of recalculation periods. There is a 'hard' lower limit on the recalculation period of about 115ms. As the period gets longer, the measured CLR becomes lower, indicative of increasing pessimism by the CAC algorithm. Similar to the decline of the measured CLR when the block length B increases, as the recalculation period increases there is a steady decline in the measured CLR.

Not only does slow performance in the calculation of the effective bandwidth cause pessimistic behaviour of the CAC algorithm, but the CAC algorithm may also be overly optimistic, causing excessive cell-loss. We would postulate that this occurs because the algorithm is using an out-of-date estimate of the effective bandwidth; by using an out-of-date estimate of the effective bandwidth the algorithm may admit a connection when the line capacity is not available. This situation, of optimistic admission causing excessive cell-loss would be more prevalent for *Hornet* where the algorithm obtains a better line utilisation by using the new connection SCR and IBT parameters when these parameters are available. *Hornet* can keep the line utilisation closer to maximum and as a result will be more adversely affected by out-of-date estimates.

In addition to differences between the *Measure* and *Hornet* performance, differences will results from the measuring system underlying the algorithm implementations. The per-connection measurements implementation of the *Measure* and *Hornet* back-end has a slower calculation period than the back-end implementation based upon aggregate measurements, as a result the per-connection CAC algorithms can be expected to have larger errors between the target CLR and the measured CLR than the aggregate method could give.

The performance of an implementation is not simply the speed at which the Equation 8 can be calculated. For the implementation based upon per-connection measurements the Table 10 reveals[3] a great deal of information about how the time is used in the estimation of an effective bandwidth. Data management involves the summing of $\tilde{X}_k$ for each connection into the single $\tilde{X}_k$ that is the input of Equation 8. This task is expensive because it requires a large amount of information be carried with each set of per-connection measurements to indicate which connections were active at the time of the measurement. Checking the validity of these samples accounts for a significant portion of the 'data management' entry in this table. The three items of this table account for 93.5% of the time calculating the effective bandwidth estimate, no other single component of the *Measure* back-end accounts for more than 0.1% of the total time. Any reduction of these components stands to return significant improvements in the speed at which estimates can be made available.

Module	% of time
Addition and Multiplication range checking (in the root solver)	65.0
Calculating the $\sum_{k=1}^{K} e^{\theta \tilde{X}_k}$ term of Equation 8	17.8
Data management	10.7

Table 10: Top three *Measure* back-end modules listed by the percentage time used in the recalculation of an estimate.

The implementation has been significantly improved since its original inception, however we still feel there is still room for improvement. In this early implementation of the *Measure* and *Hornet* algorithms there are bound to be parts of the code that can undergo further significant

[3] Inconsistencies between the results of Table 10 and those of Table 9 are still unresolved at this time but it is theorised that they are as a result of the increased memory accesses in the per connection implementation of the 'root-solver' central to *Measure*.

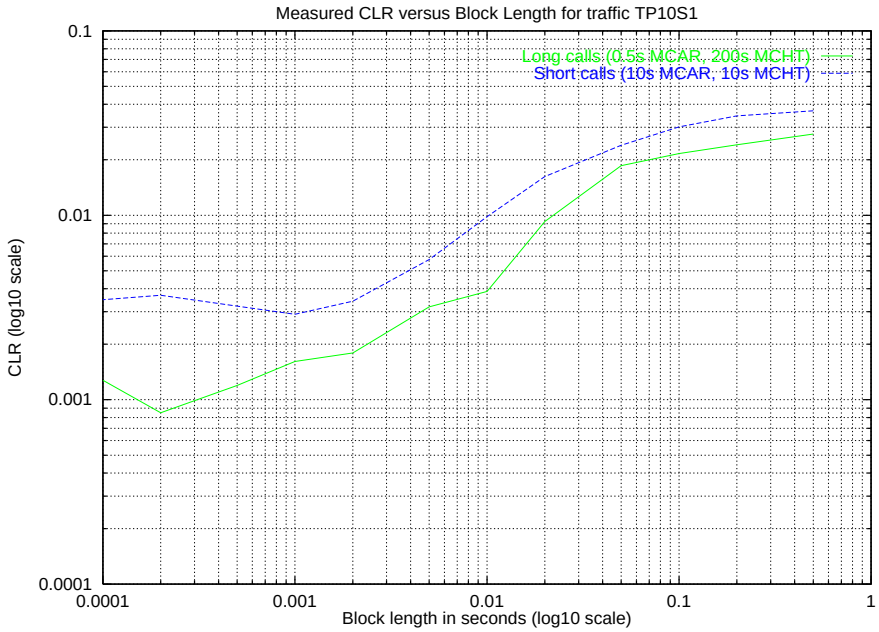


Figure 13: Measured CLR versus Block length using traffic type TP10S1. The Hornet algorithm with per-connection line statistics was used for both sets of experiments.

Figure 14: Measured CLR versus Recalculation period using traffic type TP10S1. The *Measure* algorithm with per-connection line statistics was used for this set of experiments.

optimisation. The root-solver, central to the calculation of an effective bandwidth estimate from the SCGF, is still quite primitive and it is estimated that a revised, simplification could return a five fold improvement in speed. Such an improvement would immediately return almost a magnitude improvement in the time taken to calculate estimates. However, such improvements require significant testing to ensure they are both robust and numerically sound.

Table 10 shows that for a *Measure* estimator based upon per-connection measurements a large percentage of time is spent managing the data: checking which connections are active, summing this data and so on. The *Measure* estimator based upon aggregate measurements does not have this overhead and Table 9 shows how significant the speed difference can be. While it would be unfair to conclude that the data management is alone responsible for the difference in times, the greater memory requirements for per-connection measurements would mean larger requirements for process management on the part of the operating system and as a result slower times would result.

The speed of the per-connection estimator will be directly influenced by the number of active connections at any time. The larger the number of active connections, will mean there will be greater samples to be summed to create each $\tilde{X}_k$. As a result the greater the number of concurrent connections, whether due to connections with particularly small traffic requirements, or due to the efficiency of an algorithm such as *Hornet* in admitting these connections will mean the effective bandwidth estimate will be calculated more slowly than if there were fewer connections in-progress.

In addition to the effect the number of connections has on per-connection measurments, the time taken by both per-connection and aggregate algorithms will be directly proportional to the number of blocks, K. In a very simple way: the longer the history – the longer it will take to calculate an effective bandwidth estimate.

A solution to both of these problems may be to force the implementation into hardware, thus giving a cleaner access to information about the validity of connections, the statistics of connection paths and the benefits of a fixed implementation directly in the switch. In this way we would be trading off flexibility for speed; either alternatively or as an adjunct to a direct switch implementation aggregate statistics could become a satisfactory compromise. As a result of this the management and memory overhead of per-connection statistics would be eliminated.

A solution to the problem of a potentially unbounded number of blocks the *Measure* and *Hornet* algorithm may need to deal with would be to place an upper limit on the value of K, the total number of blocks. Such a requirement would be a necessity for any production implementation where unbounded memory requirements cannot be realisticly handled. The problem that remains would be to select the maximum value of K; at first, an upper limit on the history period itself would seem appropriate. Section 6.4.1 discusses how the Measure algorithm based upon aggregate measurements uses an upper limit for the history it collects. This history limit is based upon the mean connection length, in seconds. The relationship between such a value and the value of K, the number of blocks, is the block length B. It can be seen that the problem of bounding the number of blocks, would be to place an upper limit on the value of K rather than an upper limit on the history period itself. This would have the unusual side-effect that the maximum period of time covered would change, depending upon the block length B.

Figure 15 shows an implementation of the *Measure* algorithm using aggregate line measurements and a fixed history length for each experiment. The history length is varied and this graph shows the measured CLR that results. It becomes apparent that, like the block length B, the length of history of previous blocks, is a parameter which may need to be adjusted to achieve an optimum CAC behaviour. The keeping a large number of blocks about all connections, and in particular those containing data of connections that have since left the system, has limited theoretical basis and thus a system such as the algorithms based upon aggregate measurements, with a fixed history may have an unsound foundation.

7.4 Algorithm Comparison

This Section presents results comparing the two *Measure* and two *Hornet* CAC algorithms along with *Gibbens/Kelly95 - III* a CAC from [12] and a simple *Peak Rate Allocation* policy. Results

using a number of different traffic sources are used; these traffic sources include TP10S1, VP10S4 and the traffic combination (a combination of ON-OFF sources and an ATM stream based upon the StarWars movie) that was used in the previous deliverable 3.4.1.

7.4.1 ON-OFF traffic source – TP10S1

Through the use of a traffic source that has a theoretical basis, comparison can be made between the implementation and the theory on which it has been founded; in this Section we are able to compare the results we gain by using a theoretical source with the estimations of several widely accepted estimation models. The ON-OFF traffic source TP10S1 is a 2-state ON-OFF Markov source, it is described in Section 5.1.

Table 11 presents a group of experimental results for the four variations on the *Measure* algorithm (Measure Per-connection, Measure Aggregate, Hornet Per-connection and Hornet Aggregate). These implementations and their differences are detailed at length in Section 6 but in summary those based on the *Measure* will use the declared PCR of an incoming connection as a pessimistic assumption of the load to be introduced by the new connection. In contrast *Hornet* can, in addition to the PCR parameter, make use of SCR and IBT to better estimate the worst case load a new connection may introduce to the system. The difference between Per-connection and Aggregate is that the former is based strictly on the theoretical basis of *Measure*, summing only currently active connections when calculating values of $\tilde{X}_k$ (from Equation 8. The aggregate based system uses as input aggregated line measurements and thus the traffic measurements of connections that have since been pulled-down still contribute to the value of $\tilde{X}_k$. The aggregate based system also introduces a 'history-limiting' mechanism putting an upper limit on the number of blocks that are part of the calculation process; this has the effect of discounting the data of long-since removed connections. The upper boundary on the history value is calculated using a method detailed in Section 6.4.1.

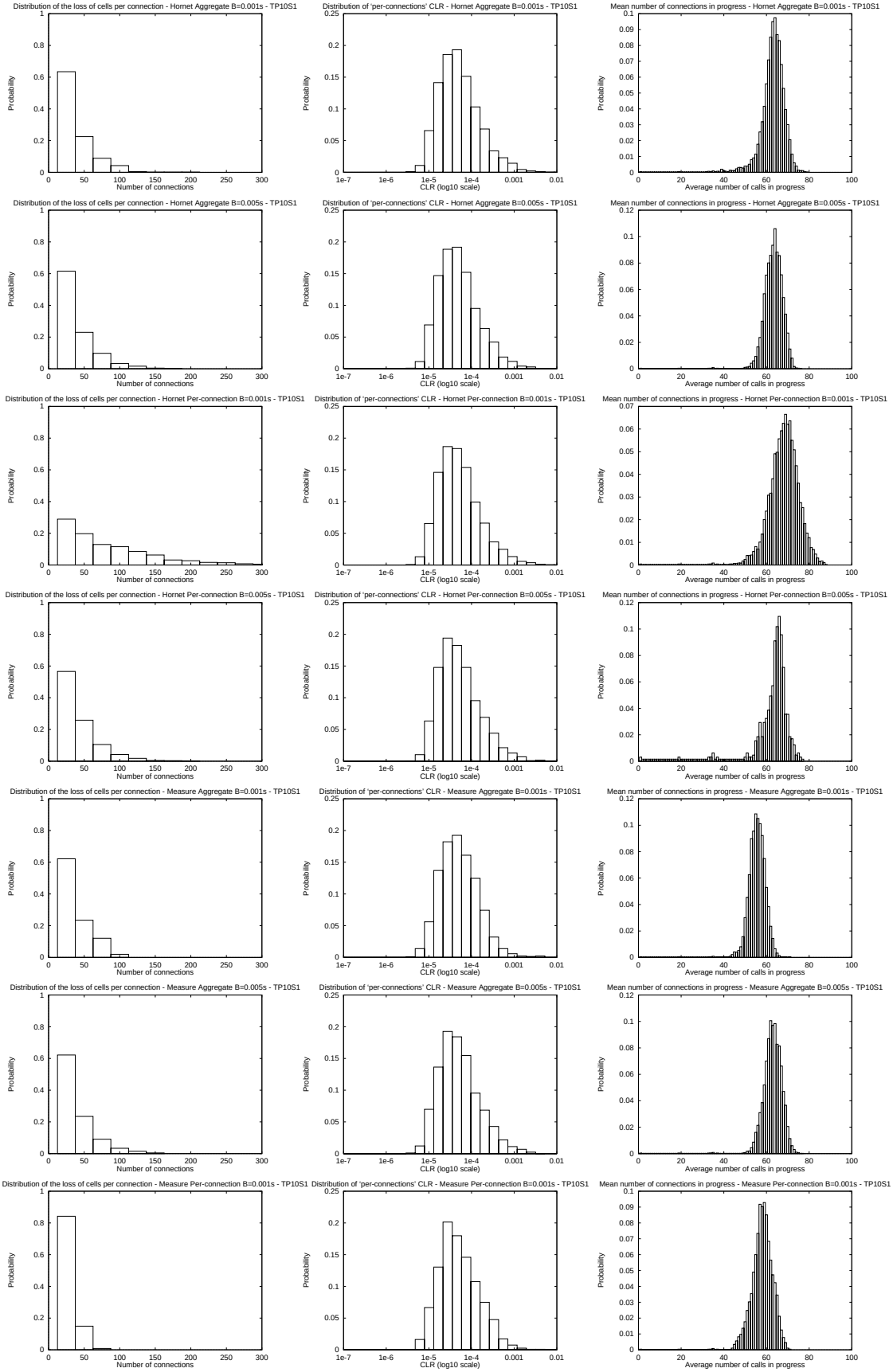
Two sets of results of experiments conducted using *Measure* and *Hornet* based CAC algorithms are presented here; these results differ in the value of block length B used. As was discussed in Section 7.2, the value of this parameter has a significant effect on the behaviour of these CAC algorithms. The results of Table 11 show that, in addition to the measured CLR, changes in the block length B will affect other aspects of the CAC operation too. Figure 16 allows the direct comparison of the profiles of the mean number of connections in progress. The difference, due to block length B, reflects the differing means, but also the distributions themselves are different too. This would support the conclusion that the block length B value extends significant control of the general behaviour of the *Measure* and *Hornet* CAC algorithms as well.

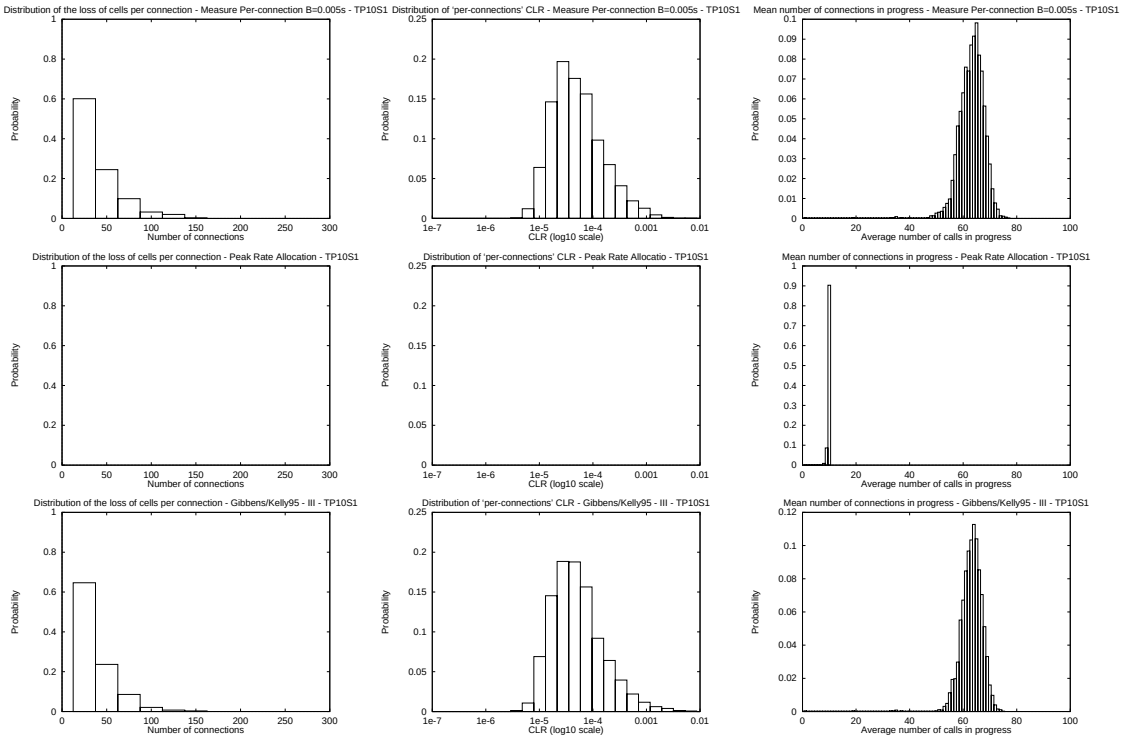
Theoretical models can also give estimates for the mean connections in progress a figure roughly comparable with the connection accept/reject ratio if the experiment has a sufficiently high MCAR. In the table an estimation for mean connections in progress is given from several theoretical estimators: Guérin [13], Elwalid [14] and Buffet & Duffield [4]. It is interesting to note the significant divergence in the results for commonly used theoretical models.

Table 12 presents three figure for each experiment of Table 11. Leftmost is the histogram of connections with a given cell loss, the centre figure is the distribution of 'per-connection' CLR and the rightmost figure is the distribution of the average number of connections admitted for a given experiment.

Table 11 reveals that the cells lost for a given measure CLR, a figure taken for the line, may not be shared evenly amount connections. The figures for *Hornet* per-connection with a block length B of 0.005s show that the line CLR is well below target (5.752×10^{-3}) but that only ≈73% of connections actually suffer a cell-loss worse than 1×10^{-3}. The appropriate figure in Table 12 clearly shows that a small number of connections are suffering significant (≈50%) cell-loss.

Table 12: Table of figures showing the distribution of cells lost per connection, the distribution of per-connection CLR and the mean number of connections in progress for the experiments summarised in Table 11.


Distribution of the loss of cells per connection - Hornet Aggregate B=0.001s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Hornet Aggregate B=0.001s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Hornet Aggregate B=0.001s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Hornet Aggregate B=0.005s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Hornet Aggregate B=0.005s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Hornet Aggregate B=0.005s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Hornet Per-connection B=0.001s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Hornet Per-connection B=0.001s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Hornet Per-connection B=0.001s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Hornet Per-connection B=0.005s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Hornet Per-connection B=0.005s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Hornet Per-connection B=0.005s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Measure Aggregate B=0.001s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Measure Aggregate B=0.001s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Measure Aggregate B=0.001s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Measure Aggregate B=0.005s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Measure Aggregate B=0.005s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Measure Aggregate B=0.005s - TP10S1
Probability
Average number of calls in progress
Distribution of the loss of cells per connection - Measure Per-connection B=0.001s - TP10S1
Probability
Number of connections
Distribution of 'per-connections' CLR - Measure Per-connection B=0.001s - TP10S1
Probability
CLR (log10 scale)
Mean number of connections in progress - Measure Per-connection B=0.001s - TP10S1
Probability
Average number of calls in progress



7.4.2 Video traffic source – VP10S4

The experimental environment we use, described in Section 4, allows us to generate connections carrying real streams of traffic. Such real traffic streams could be pre-recorded from the original source, or they could be summarised and this summary (of the video characteristics such as frame size, frame rate, etc.) then used to create traffic on-demand that is identical to the traffic streams originating from sources. Such a facility has enabled us to record the carriage of the video stream through the ATM network. This record of the network activity can be replayed by the experimental environment, on demand, per connection. The result is we are able to simulate a network with every connection carrying part or all of the recorded video session. Traffic stream VP10S4 is further detailed in Section 5.3, it has been used in this Section of the study to allow the comparison of the traffic streams with the carriage of real video data through the network.

Table 13 presents the results of a set of experiments conducted with the VP10S4 traffic source. In the same manner as Table 11 several results for the *Measure* and *Hornet* algorithms have been given; these include two different values of the block length B and *Measure* and *Hornet* based upon two different measurement techniques: Per-connection and Aggregate line measurements. Section 6 gives full details on how the various CAC algorithms differ.

In addition to the *Measure* and *Hornet* algorithms; we present results gained using *Gibbens/Kelly95 - III* a CAC from [12] as well as a simple *Peak Rate Allocation* policy. Alongside these practical results, the estimations gained from several popular theoretical models have also been given. In the table an estimation for mean connections in progress is given from several theoretical estimators: Guérin [13], Elwalid [14] and Buffet & Duffield [4]. Once again, significant divergence in the results is present for commonly used theoretical models. This divergence is most likely due to the video stream not fitting as closely the 2-state Poisson traffic as these theoretical models assume. This failure of Poisson traffic models to represent real-world data is a well known problem and has lead to numerous papers most notably [15].

In comparison with CAC algorithms based upon these estimations, we have maintained that "real-world" traffic will not present the same challenge to *Measure* based estimation. Table 13

Algorithm	CLR for a 100 cell buffer cell loss	Prob. calls with CLR $> 1 \times 10^{-3}$	Prob. call has	Connection call has in progress	Mean connections	Mean line Utilisation
Measure Per-connection	3.780×10^{-4}	4.954×10^{-1}	7.533×10^{-2}	0.591	57.726	0.575
Measure Aggregate	1.702×10^{-4}	3.029×10^{-1}	1.833×10^{-2}	0.559	55.387	0.553
Hornet Per-connection	2.903×10^{-3}	8.600×10^{-1}	8.030×10^{-1}	0.680	67.759	0.670
Hornet Aggregate	9.537×10^{-4}	7.034×10^{-1}	3.612×10^{-1}	0.639	62.658	0.626
Block length $B = 1 \times 10^{-3}$						
Measure Per-connection	9.447×10^{-4}	7.027×10^{-1}	3.498×10^{-1}	0.638	62.871	0.630
Measure Aggregate	8.918×10^{-4}	6.919×10^{-1}	3.278×10^{-1}	0.635	62.679	0.625
Hornet Per-connection	5.752×10^{-3}	7.313×10^{-1}	4.323×10^{-1}	0.740	73.181	0.726
Hornet Aggregate	8.715×10^{-4}	7.027×10^{-1}	3.498×10^{-1}	0.635	62.55	0.624
Block length $B = 5 \times 10^{-3}$						
Peak Rate Allocation	0	0	0	0.100	9.89	0.080
Gibbens/Kelly95 - III scalar $s = 2$	8.761×10^{-4}	6.960×10^{-1}	3.161×10^{-1}	0.637	62.958	0.630
Theoretical model estimations						
Guérin [13]	1.00×10^{-3}				36.92	
Elwalid [14]	1.00×10^{-3}				42.98	
Buffett & Dufield [4]	1.00×10^{-3}				45.20	

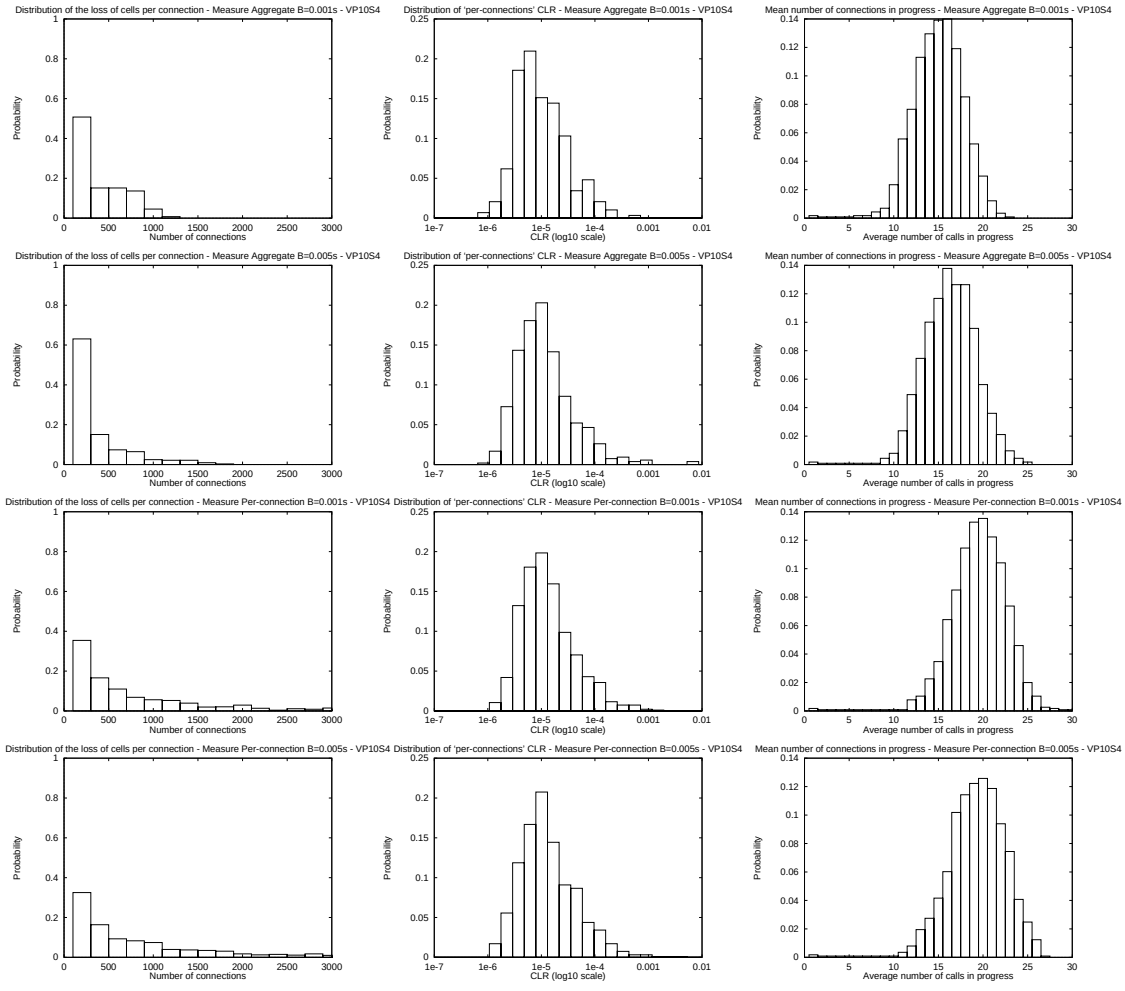
Table 11: Results for traffic source TP10S1

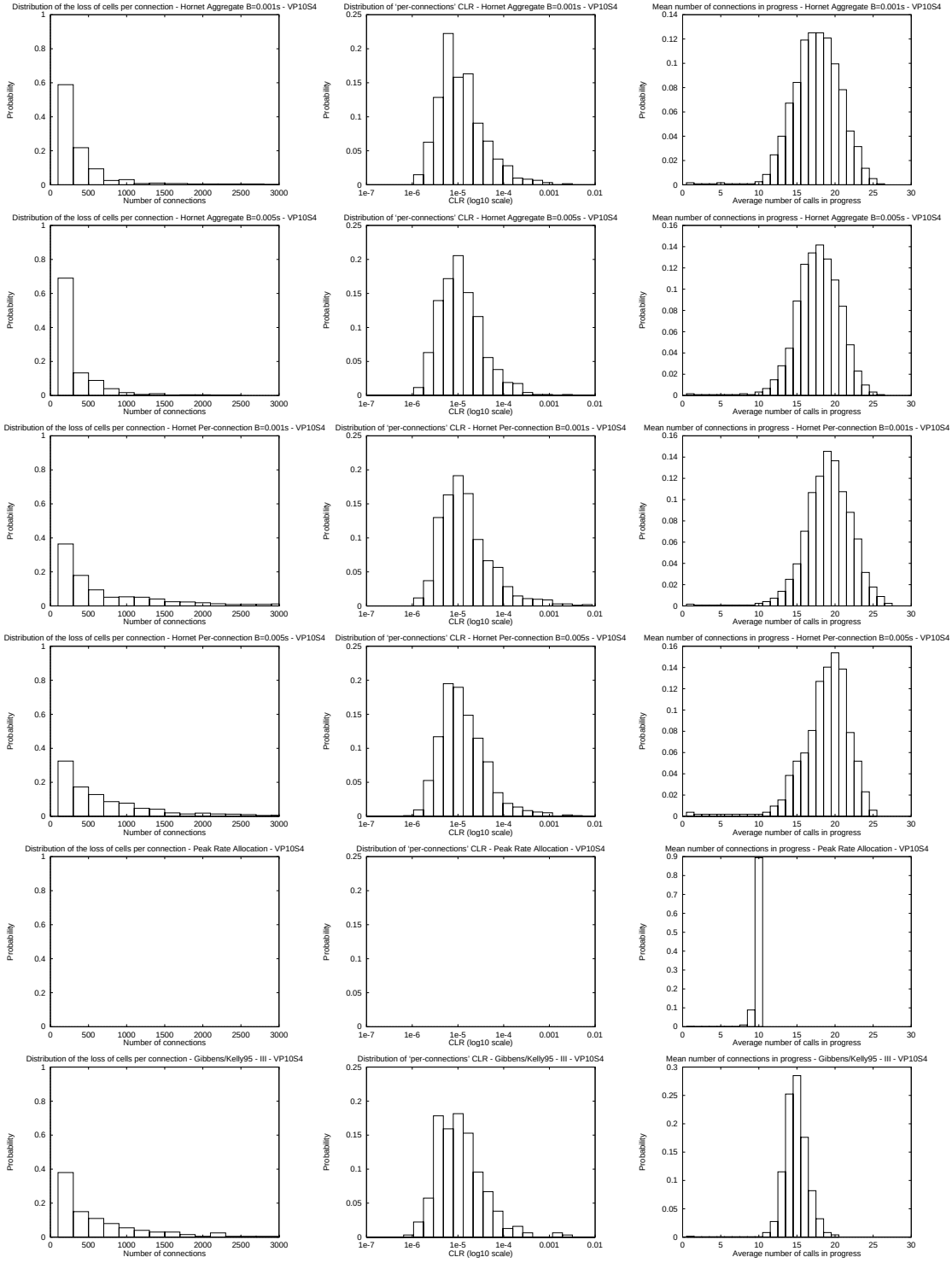
results bear this out, although these results do show the importance of selecting appropriate values for block length B.

Figure 17 illustrates that the range of CAC algorithms that give different acceptance ratios will also result in different profiles of 'connections in progress.' Apart from the results for the Peak Rate Allocation policy, the three other sets of results were selected to be illustrated together as the measured CLR for each is near identical. Such results allow us to very quickly ascertain the more effective algorithm (assuming other factors such as measured CLR are identical). In this figure we note that the *Hornet* and *Measure* results present a near identical profile of 'connections in progress' and that both have significantly more connections in progress than the *Gibbens/Kelly95 - III* CAC algorithm.

In a fashion similar to that of Table 12, Table 14 presents three figure for each experiment of Table 13. Leftmost is the histogram of connections with a given cell loss, the centre figure is the distribution of 'per-connection' CLR and the rightmost figure is the distribution of the average number of connections admitted for a given experiment.

Table 14: Table of figures showing the distribution of cells lost per connection, the distribution of per-connection CLR and Mean number of connections in progress for the experiments summarised in Table 13.





7.4.3 Traffic Mix

Previous Sections have reported on the behaviour of *Measure* and *Hornet* when used with connections carrying a single traffic source; this Section reports on results gained when the connections carry mixtures of traffic.

Algorithm	CLR for a 100 cell buffer	Prob. call has cell loss	Prob. call has CLR $> 1 \times 10^{-3}$	Connection accept ratio	Mean connections in progress	Mean line Utilisation
Measure Per-connection	6.279×10^{-3}	8.441×10^{-1}	7.272×10^{-1}	0.195	19.185	0.753
Measure Aggregate	4.523×10^{-4}	2.568×10^{-1}	9.091×10^{-2}	0.146	14.606	0.509
Hornet Per-connection	6.597×10^{-3}	8.428×10^{-1}	7.177×10^{-1}	0.196	18.783	0.703
Hornet Aggregate	1.256×10^{-3}	5.251×10^{-1}	2.578×10^{-1}	0.176	17.367	0.670
Block length $B = 1 \times 10^{-3}$						
Measure Per-connection	6.565×10^{-3}	8.431×10^{-1}	7.259×10^{-1}	0.191	18.921	0.754
Measure Aggregate	9.957×10^{-4}	4.786×10^{-1}	2.103×10^{-1}	0.161	15.861	0.653
Hornet Per-connection	4.767×10^{-3}	8.160×10^{-1}	6.758×10^{-1}	0.185	18.478	0.688
Hornet Aggregate	1.228×10^{-3}	5.684×10^{-1}	2.813×10^{-1}	0.182	17.499	0.685
Block length $B = 5 \times 10^{-3}$						
Peak Rate Allocation	0	0	0	0.101	9.885	0.389
Gibbens/Kelly95 - III scalar $s = 15$	1.433×10^{-3}	2.584×10^{-1}	1.366×10^{-1}	0.157	14.765	0.578
Theoretical model estimations						
Guérin [13]	1.00×10^{-3}		15.12			
Elwalid [14]	1.00×10^{-3}		15.65			
Buffett & Dufield [4]	1.00×10^{-3}		17.12			

Table 13: Results for traffic source VP10S4

In an attempt to replicate in practice results first created in simulation, we tested the algorithms against a combination of traffic and model of connection attempts that was used in Section 5 Simulation Results, of the previous deliverable 3.4.1.

For this set of experiments a combination of ON-OFF theoretical model traffic and traffic based on video data were used, the video traffic used was based upon the StarWars movie, the details of the traffic used is in Section 5.2. The ON-OFF two-state Markovian source, T2MIX, is detailed in Section 5.4.

In an attempt to replicate the previous deliverable as closely as possible we set the maximum buffer size to 500 cells and we used a CLR constraint of 1×10^{-4} for all of the results with this traffic mix. We also adopted the same model of connection attempts as had been used in the previous deliverable; we have used connection lengths which are exponentially distributed. Connection attempts have an exponentially distributed length; long connections have a mean length of 60 seconds, and short connections a mean of 10 seconds. Both long and short duration connections have a mean connection arrival rate of 5 connection attempts per second.

Each accepted connection transmits an independent *trace*. In the case of Star Wars, the trace is derived by randomly selecting a start point in the traffic trace. For the ON-OFF source, each connection chooses its parameters randomly: the mean (as a fraction of the peak) is chosen from 0.1, 0.3, 0.5, 0.7, and 0.9 with probabilities 1/16, 4/16, 6/16, 4/16, and 1/16 respectively. The burstiness is chosen from 0.3, 0.5, 0.7, and 0.9 with probabilities 1/8, 3/8, 3/8, and 1/8 respectively. Connections are not correlated in any way, although the *same random seed* was used with each different experiment, resulting in precisely the same connection arrivals process in each case. This allows us to compare not only the statistical properties of the algorithms, but also their dynamic behaviour.

Table 15 presents a set of the results obtained for the mixed traffic connections. The results are very disappointing with both *Measure* and *Hornet* per-connection algorithms over admitting connections and thus causing a large value of measured CLR. While the reasons for these results have not been fully established; several plausible reasons are suggested.

Algorithm	CLR for a 500 cell buffer	Connection accept ratio	Mean connections in progress	Mean line Utilisation
Measure Per-connection	1.237×10^{-1}	0.061	20.077	1.000
Hornet Per-connection	1.121×10^{-1}	0.064	20.020	1.000
Block length $B = 2 \times 10^{-3}$				
Peak Rate Allocation	0	0.027	8.565	0.521
Gibbens/Kelly95 - III	3.586×10^{-2}	0.039	13.857	1.000
scalar $s = 2$				

Table 15: Results for mixed traffic experiments.

Initially we were concerned that the nature of the sources themselves were the cause for the poor behaviour; while video sources have not presented such unusually poor behaviour previously, the traffic source T2MIX used in this set of experiments were less "bursty" than TP10S1. As well as having a lower activity and smaller mean burst lengths, the connections generally high SCR. It was thought that any of these factors could have contributed to the poor performance. One approach was to run experiments with all parameters (target CLR, block length B, buffer size) held constant but to only use one traffic type at a time. Table 16 details the results obtained for these experiments. Previously used traffic source, TP10S1, is included for comparison.

In comparison to the TP10S1 source, both of the other traffic sources perform far worse. The mean holding time appears to have no effect on the performance for each individual traffic stream. Firstly, we already have established that the recalculation speed of the per-connection *Measure* and *Hornet* algorithms is sub-optimal, it is easy to envisage the effect that slow and incomplete results may cause and we have discussed this in general terms in Section 7.3. Secondly, the method

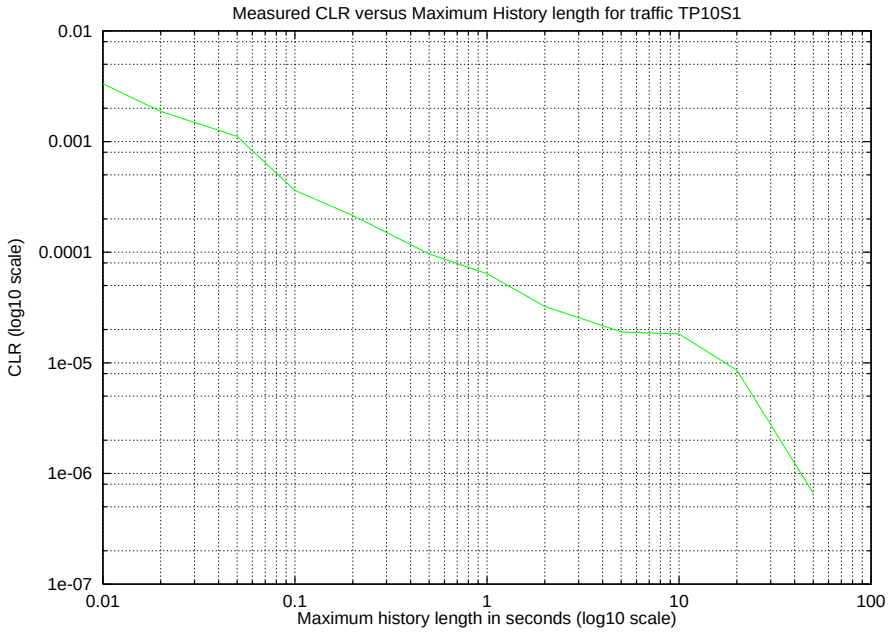


Figure 15: Measured CLR versus history length using traffic type TP10S1. The *Measure* algorithm with aggregate line statistics was used for this set of experiments.

Traffic	Mean connection hold time	CLR for a 500 cell buffer	Connection accept ratio	Mean connections in progress	Mean line Utilisation
StarWars	10	0.411	0.133	11.961	0.990
StarWars	60	0.045	0.031	12.317	1.000
T2MIX	10	0.111	0.197	19.488	1.000
T2MIX	60	0.108	0.042	20.128	1.000
TP10S1	10	0.000285	0.794	77.865	0.777

Table 16: Results for individual components of mixed traffic experiments.

of selection of the value of block length B still requires some work and as a result the poor results could be due to selection of this parameter. On the basis that the results for TP10S1 have the measured CLR approaching the target CLR, we are forced to conclude that the block length B used for these experiments is not suitable for the T2MIX and StarWars traffic sources.

8 Conclusions

Interest in CAC algorithms stems from the need for a network user and a network provider to establish an agreement on the QoS for a connection the user wishes to have admitted into the network. This study has presented results of experiments culminating in the evaluation of the *Measure* CAC algorithm and the comparison of that CAC algorithm with several other CAC algorithms.

Section 2 and Section 3 established the terminology and presents the theoretical framework of *Measure* and described how the theoretical foundations of *Measure* can be applied to develop algorithms for connection admission control.

Section 4 detailed the experimental environment used in the evaluation of the Measure CAC algorithm. Of particular interest was the implementation of *Measure* into two components, back-end and front-end. The delineation of front and back-end is coincidently the delineation between predictor and estimator in the *Measure* algorithm.

Section 5 discussed the two types of sources used in this CAC evaluation. One source being based upon a theoretical model and the other derived from a stream of video data,

Section 6 describes specific implementation issues for the *Measure* and *Hornet* algorithms. One aspect of the implementation was how the delineation between back-end and front-end also corresponded with division in the algorithm between estimator and predictor. This section also covered the mechanism that implements a boundary on the history of data processed by *Measure*.

Section 7 reports results for the experiments we have conducted. A comparison was conducted between implementations of the *Measure* algorithm comparing measured CLR with target CLR. This revealed highly variable performance. The parameters of *Measure* were then compared with their performance and this showed the importantance in selecting block length B. Performance results indicated the significant differences between per-connection and aggregate measurements as well as illustrating how the speed of recalculation affects the algorithms operation. Finally this section detailed a comparison between the *Measure* and *Hornet* algorithms along with several other CAC mechanisms. These results reinforced the importance of selecting block length B and the importance of speedy recalculations of results.

Future Work

The scope for future work is very broad, and there are a number of areas of potential development. At this stage we see considerable scope exists to further investigate the relationship between the block length B and traffic characteristics: both of the traffic and of the connection profile.

With the potential engineering difficulties associated with per-connection based measurements, many aspects of *Measure* based upon aggregate measurements deserve future attention.

Additionally, because of the importance of a speedy algorithm, future work could assess the impact of a less accurate algorithm if it is able to return results more speedily.

References

[1] M. W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar vbr video traffic. In *Proceedings ACM SIGCOMM 94*, pages 269–280, London, UK, August 1994.

[2] Matthias Grossglauser and Jean-Chrysostome Bolot. On the relevance of long-range dependence in network traffic. In *Proceedings ACM Sigcomm*, Stanford University CA, August 1996.

[3] Richard Black, Ian Leslie, and Derek McAuley. Experiences of building an ATM switch for the Local Area. In *Proceedings ACM SIGCOMM*, volume 24(4), September 1994.

[4] E. Buffet and N.G. Duffield. Exponential upper bounds via martingales for multiplexers with markovian arrivals. *J. Appl. Prob.*, 31:1049–1061, 1992.

[5] Simon Crosby, Ian Leslie, John Lewis, Raymond Russell, Meriel Huggard, and Brian McGurk. Predicting effective bandwidths of ATM and ethernet traffic. In *Proceedings Thirteenth UK Teletraffic Symposium*, Strathclyde University, Glasgow, March 1996. IEE.

[6] Joseph Y. Hui. Resource allocation for broadband networks. *IEEE JSAC*, 6(9):1598–1608, December 1988.

[7] Richard Black and Simon Crosby. Experience and results from the implementation of an ATM socket family. In *Proceedings USENIX Winter Technical Conference*, San Francisco, January 1994.

[8] *ATM Document Collection.* Systems Research Group, University of Cambridge Computer Laboratory, 3rd (The Blue Book) edition, March 1994. SRG Technical Note available as: http://www.cl.cam.ac.uk/Research/SRG/bluebook.html.

[9] Song Chong, San-qi Li, and Joydeep Ghosh. Dynamic bandwidth allocation for efficient transport of real-time VBR video over ATM. In *Proceedings IEEE INFOCOM*, pages 81–90, Toronto, Ontario, Canada, June 1994.

[10] Simon Crosby, Ian Leslie, John Lewis, Neil O'Connell, Raymond Russell, and Fergal Toomey. Bypassing modelling: an investigation of entropy as a traffic descriptor in the Fairisle ATM network. In *Proceedings Twelfth UK Teletraffic Symposium*, Windsor, England, March 1995. IEE, Springer Verlag.

[11] Matthias Grossglauser and David Tse. A framework for robust measurement-based admission control. *ACM SIGCOMM Computer Communications Review*, 27(4):237–248, Oct 1997.

[12] R. J. Gibbens and F. P. Kelly. Measurement-based connection admission control. In *15th International Teletraffic Congress Procesdings*, June 1997.

[13] R. Guerin, H. Ahmadi, and Naghshineh. Equivalent capacity and its application to bandwidth allocation in high speed networks. *IEEE JSAC*, 9:968–981, 1991.

[14] A.I. Elwalid, D. Mitra, and T.E. Stern. Statistical multiplexing of Markov modulated sources: Theory and computational algorithms. In *Proceedings of the 13th International Teletraffic Congress*, Copenhagen, June 1991.

[15] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. In *Proceedings ACM SIGCOMM 94*, pages 257–268, London, UK, August 1994.

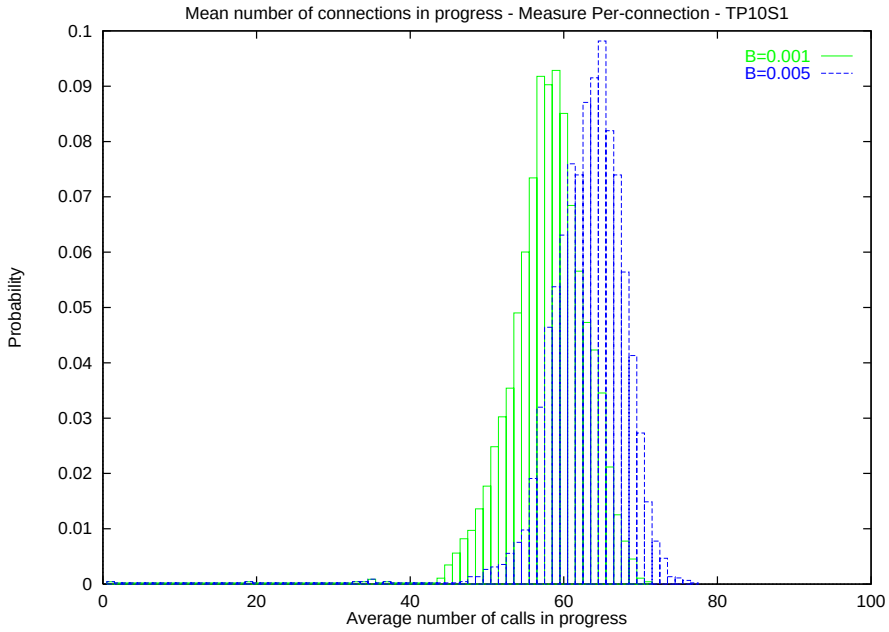


Figure 16: A comparison of the mean number of connections in progress for Measure per-connection using different values of block length.

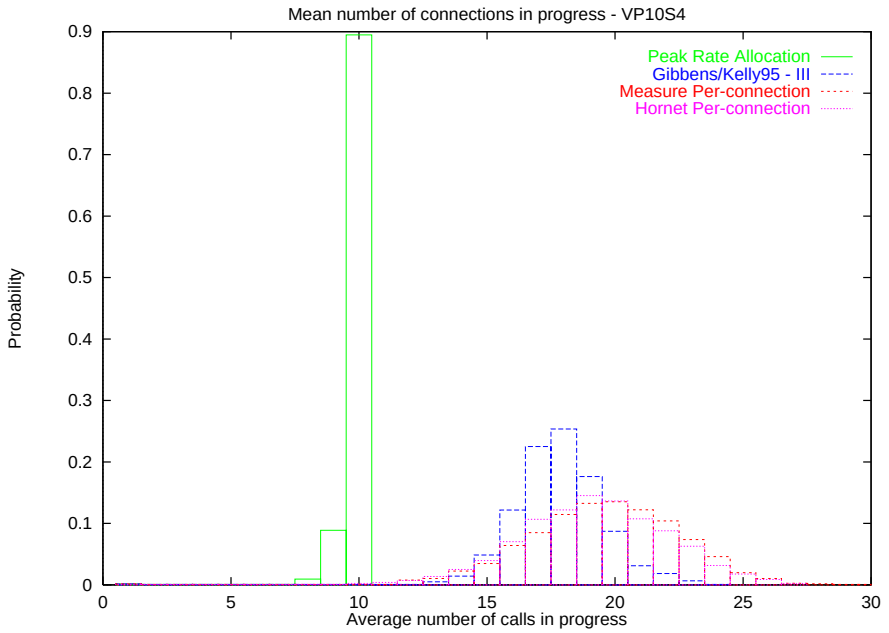


Figure 17: A comparison of the distribution of the mean number of 'connections in progress,' using a number of CAC algorithms.