ABSTRACT

Hacıömeroğlu, Fatih. On-line Measurement-based Capacity Allocation Schemes (Under the direction of Dr. Michael Devetsikiotis).

Today's high-speed packet-switched networks are faced with the task of handling an increasing amount and variety of services, requiring different QoS constraints. To cope with this demand, the networks need dynamic and measurement-based resource allocation algorithms. For this task, the choice of appropriately accurate but also *practically implementable* algorithms is crucial.

In this thesis, we perform a comparative study of alternative on-line algorithms, we analyze their complexity, and perform comparisons via simulation experiments. Our motivation is to use these algorithms in the data plane of "self-sizing" frameworks, and make use of their output in taking control plane decisions either locally or globally, in an *on-line* fashion.

Due to the dynamic characteristics of the algorithms, we encounter the choice of time resolution, namely the setting of measurement time scale and window. After numerous simulations, we gain insight on the critical effect of these choices on the performance of the algorithms. We deduce that the time scale parameter itself is to be determined *dynamically* so that measurement-based algorithms can perform successfully independent from the varying traffic conditions. Finally, we demonstrate the effectiveness of this new approach over the static one, in our measurement-based capacity allocation algorithms.

On-line Measurement-based Capacity Allocation Schemes

by

Fatih Hacıömeroğlu

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Electrical and Computer Engineering

Raleigh

2003

Approved By:

Dr. George N. Rouskas

Dr. J. Keith Townsend

Dr. Michael Devetsikiotis
Chair of Advisory Committee

To my family...

Biography

Fatih Hacıömeroğlu was born on October 5, 1977 in Trabzon, Turkey. He grew up in Ankara. He was admitted to the Middle East Technical University in 1995, and earned his Bachelor of Science degree in Electrical and Electronics Engineering in June 2000. He pursued a Masters degree outside Turkey to learn more about other cultures while broadening his knowledge in developments in technology. He joined the graduate program in Electrical and Computer Engineering at North Carolina State University in August 2000.

Acknowledgements

I would like to express my deepest appreciation to my advisor Dr. Michael Devetsikiotis for all his support, encouragement, availability and invaluable guidance throughout this research. I would like to thank my committee members, Dr. George N. Rouskas and Dr. J. Keith Townsend, for taking interest in my work. I would like to take this opportunity to thank Center of Advanced Computer Communication (CACC) and Alcatel in Plato for the financial support they provided and for the research direction they outlined.

I would like to give my appreciation to my fellow graduate students, Peng Xu, Vladica Stanisic, Srikant Nalatwad and Sai Oruganti for their kindness and friendship. I especially want to thank Bob Callaway for sharing his implementation results which constitutes chapter 4. Very special thanks to my roommates Ege Yildizoglu, Mustafa Bakkal and Engin Murat Reis for their patience and support. Thank you for the laughter, smiles and friendship. I wish you best luck in the rest of your lives.

One thing for sure, I can not thank enough my parents and all my family for their constant encouragement and belief in me. You have always been unbelievable. You were my biggest support in this adventure. Finally, I would like to express my very special thanks to my sister Sabire, who has always been my relief in my most difficult times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The demand on high-speed networks is becoming higher and tougher to satisfy everyday, with the invention and commercialization of new bandwidth-hungry applications. Network resources need to be increased accordingly, to sustain an acceptable level of service. However, it is not always feasible to increase resources at the same pace of the increase in the traffic demand. In this regard, the bandwidth allocation in high-speed QoS-oriented networks is critical and needs to be made dynamic, adaptive and measurement-based, rather than static, to attain a more efficient use of resources. Especially for network links shared through statistical multiplexing, adaptive bandwidth allocation algorithms based on traffic measurements can achieve important gains.

In this context, it is very important to choose measurement methods that satisfy stringent constraints in terms of both accuracy and complexity. In this thesis, first, we identified practically-implementable dynamic capacity allocation algorithms, which are based only on measurements instead of unreasonable assumptions about the incoming traffic. Then, after performing their analytical comparisons, we selected promising ones for further simulation-based performance comparisons. We also tested the algorithms in a real network experiment and show the applicability of the algorithms into practical world. In the light of the simulation results, we deduced the critical effect of measurement time scales on the performance. Finally, we proposed an approach overcoming this dependency by adjusting the measurement time scale dynamically.

1.1 Motivation

The algorithms in this thesis have the potential to be used in the data plane of self-sizing network frameworks such as [26, 10, 27] and in which every node in the network runs a measurement-based bandwidth allocation algorithm for every traffic class or "band". Periodically, the output of the algorithms, which give the required capacity demands of the traffic types, are collected, and either locally or globally, a control plane action is taken (i.e., the virtual links or scheduling allocations are re-calculated) so as to minimize a predefined objective such as bandwidth cost or maximize revenue.

Our purpose is to obtain a feasible algorithm which is able to use bandwidth *optimally* while still obtaining a performance close to the QoS target. The ideal algorithm should not require any knowledge or unreasonable assumptions on the traffic, and be based completely on measurements.

1.2 Background

Most of the algorithms in this thesis naturally originated from the effective bandwidth concept, since effective bandwidth is the amount of the required bandwidth to be allocated for the satisfaction of a QoS constraint. Furthermore, in the literature, we identified two research areas which are related to our aim. These are network traffic prediction [17, 19, 12, 7] and measurement-based admission control (MBAC) [1, 4, 20].

MBAC algorithms are composed of separate measurement procedure and admission criterion. We investigated the applicability of these separable measurement procedures for our purposes. However, other than the trivial Gaussian Approximation bandwidth estimator, such methods are not suitable for on-line re-sizing due to the fact that either they rely on unreasonable information (i.e., they presume that a priori traffic descriptors, such as present number of connections are known) or they have unacceptable computational complexity.

These incompatibilities stem from the different design considerations between measurement-based estimators in MBAC and *self-sizing* frameworks. First, MBACs are designed to operate only in the ingress nodes, where admission decisions are taken. Second, the period of execution of MBAC algorithms is at the *connection level* time scales. Our aim is to obtain *on-line* algorithms, working on every node in the network. Therefore their

timescale and computational complexity are smaller than connection level timescales. Similar to MBAC algorithms, traffic predictors do not suit our consideration either. The reason this time is not because they are centralized and computationally complex as in MBAC algorithms, but that they do not target a QoS constraint.

1.3 Contribution

First, despite several specific techniques, such a comprehensive comparison study, under the aim of on-line resource allocation, has not been carried out previously. And more important than comparison, this thesis addresses the measurement time scale problem, which is present in any measurement-based algorithm.

To illustrate, we observed that the performance of the algorithms change drastically, when measurements are taken every 0.01s. instead of every 1s. Moreover, the specific measurement time values are *relative* to the input traffic, i.e. 0.01s. can be a good sampling interval for traffic A, but 1s. can be a good sampling interval for another traffic B. However, our aim was to obtain an independent algorithm, not based on a priori traffic knowledge. Therefore, instead of taking a static sampling interval, we calculated and adjusted the sampling interval based on traffic measurements themselves.

In short, besides comparing on-line capacity allocation algorithms to be implemented in real-switches, this thesis points out the weakness of using static measurement time scale, and shows the robustness of using dynamic measurement time scale under varying traffic conditions.

1.4 Outline

The remainder of the thesis is structured as follows. Chapter 2 defines several measurement-based on-line capacity allocation algorithms and discuss their practical implementation issues such as computational complexities, accuracies and performance responses against different network conditions. Table 2.1 summarizes the main outcomes of this comparison study. We select three promising algorithms. Under the simulation scenario of Figure 3.1, we investigate and evaluate their performances in chapter 3. In addition to performance evaluation, we aim to observe the response of the algorithms against the

measurement time scale choices. Chapter 4 exhibits a practical implementation, in which our algorithms are programmed in an edge router and dimensions the ingress traffic coming into the network by performing on-line link sizing. Chapter 5 proposes a dynamic (i.e. measurement-based) measurement time scale, instead of a static one. Section 5.2 discusses the relevance of this approach. Table 5.1 demonstrates the resulting performance robustness. Finally, we conclude with a discussion of the results and outline prospects for further enhancements.

Chapter 2

Measurement-Based On-line Capacity Allocation Algorithms

2.1 Introduction

Our literature survey of on-line measurement-based on-line capacity allocation algorithms resulted in the algorithms we present in this chapter. Each algorithm is introduced in its separate section in the following organizational order. First the references and the capacity allocation formula are provided. Then the issues regarding the estimation of the formula parameters are discussed, including their computation and memory complexity. After presenting the algorithms, their comparison is performed in section 2.3, where the promising ones are selected to be analyzed further in chapter 3.

2.2 Algorithms

The algorithms in this thesis take a window of traffic measurements as input, estimate the parameters they need in a bandwidth allocation calculation formula and output the required amount of capacity to be reallocated.

The measurements correspond to the amount of the incoming traffic during a *slot*

duration, t_{slot}. Consequently, the algorithm is called periodically every $N * t_{slot}$ seconds (the reallocation period), where N is the window size. The effects of the choices of t_{slot} and N on the algorithm's performance are investigated later in the thesis, through the simulation study given in chapter 3.

2.2.1 Direct EB Allocation (DEB)

When the subject is resource allocation, the first notion that comes to mind is the effective bandwidth concept [13]. The Direct Effective Bandwidth Allocation algorithm relies on the direct analytical evaluation of the effective bandwidth formula.

$$eb(s,t) = \frac{\ln(E(e^{sX(0,t)}))}{st} \tag{2.1}$$

$X(0,t)$ is the amount of incoming work during a duration of t. The (s,t) parameters are the so-called *space* and *time* parameters. They characterize the link's operating point and depend on the context of the stream, i.e, link resources and the characteristics of the multiplexed traffic. The space parameter s shows the degree of multiplexing of the link and the degree of QoS requirement. If QoS requirements become looser, or if the degree of multiplexing increases, s tends to zero and the effective bandwidth approaches the mean rate. If QoS requirements becomes tighter, or if the degree of multiplexing decreases, s tends to infinity and effective bandwidth of the source approaches maximum rate of $max(X(0,t))/t$, measured over an interval t. The time parameter t corresponds to the most probable duration of buffer busy period prior to overflow.

The effective bandwidth concept is related to the Large Deviation Theory in the following way. QoS goals are usually loss probability and delay requirements. The delay requirement is usually satisfied by properly adjusting buffer sizes, and the loss probability is assumed to be equal to buffer overflow probability. This assumption is widely accepted to be valid. Although a counter example can be given as follows, if traffic is composed of periodic spikes of batch arrivals (where a batch load is much more than buffer size), loss probability is close to 1, whereas buffer overflow probability is close to 0. But this is an extreme condition, and almost always, buffer overflow probability is taken as equal to loss probability. Therefore, to satisfy a loss probability constraint, statistical calculation of buffer overflow probability is analyzed. The analysis turns out to be the calculation of

the tail probability of the queue distribution. As the loss probability values are very small, the Large Deviation Theory (LDT), which deals with rare event probabilities, is suitably applied in this effective bandwidth problem.

Unlike the estimation of observable parameters such as the mean and variance, the s parameter can not be directly estimated from the measurements. The space parameter is calculated by using Large Deviations Theory (LDT) and by making a large buffer assumption (LBA). LDT deals with rare event probabilities and is suitably applied to the effective bandwidth problem since loss probability constraints to be satisfied are very small. The loss probability in a buffer of size B is approximated by the probability that queue content exceeds threshold B in an infinite (or large) buffer (LBA). In large deviations analysis, the overflow probability is calculated from an asymptotically exponential decrease assumption (2.2), where s is the space parameter, being a function of the server capacity, C.

$$P(B < Q) = e^{-s(C)B} \tag{2.2}$$

The space parameter s is found from (2.2), and defined as the working point of the buffer. Having calculated s, the time parameter t and the expectation remains to be calculated from measurements. The t parameter is actually the slot duration, the period with which the measurements are taken, and it is a set parameter, and does not need estimation. Time parameter is related to time scales, which are responsible for buffer overflow. It should be chosen small enough so that traffic is observed for buffer overflow analysis. Finally, the expectation in (2.1) is approximated by a time average, as suggested in [14].

The empirical evaluation of (2.1) is simulated and compared with analytical effective bandwidth of known Poisson and On-Off source types in [24]. It is shown that the empirical effective bandwidth estimation is quite accurate for On-Off traffic, whereas accuracy in Poisson traffic depends on the space parameter. However, it is also shown that for the interested, realistic region of the space parameter, empirical estimation using the direct estimator is accurate. Moreover, this method, relying on direct evaluation of the effective bandwidth formula has computational and memory complexity of $O(N)$. That is they are directly proportional to the size of the measurement window (N). But note that, this method is not robust against long range dependent traffic and amplitude of traffic, since they may cause convergence and numerical overflow problems, respectively.

2.2.2 Courcoubetis EB Allocation (CEB)

[3] provides another effective bandwidth formula (2.3) based on Large Deviation Theory and large buffer assumption, similar to the assumptions in the Direct Effective Bandwidth Allocation algorithm.

$$eb = m + \frac{ID\, s}{2B} \tag{2.3}$$

The parameters m, B, s and ID are the mean rate, buffer size, space parameter and index of dispersion of $X[0,t]$. The space parameter s is calculated as in the first algorithm, using (2.2).

The index of dispersion parameter shows the variability of a process over different time scales. Ideally, complete characterization of the underlying traffic arrival process is needed for its exact calculation. We estimate it from (2.4), using the meaningful autocorrelation values falling into the measurement window N (note that only autocorrelations with lags less than $N/4$ are used, to eliminate statistical insignificancy). Due to the measurements of autocorrelations in the measurement window, the computational complexity is proportional to N^2.

$$ID = Var(X(0,t)) \left(1 + 2 \sum_{k=1}^{1/4\,N} \left(1 - 4\,\frac{k-1}{N} \right) AC(k) \right) \left(\sum_{k=1}^{N} \frac{X(0,t)}{N} \right)^{-1} \tag{2.4}$$

2.2.3 Many Sources Asymptotic EB Allocation (MSAEB)

This approach is also based on the effective bandwidth approach similar to the first two algorithms, but unlike them, this algorithm uses a different way of estimating the time and space parameters in (2.1) as described in [2]. Many Sources Assumption is made, instead of Large Buffer Assumption, while using Large Deviations Theory (LDT) to solve the problem of estimation of the space and time parameters (s,t).

If M sources are multiplexed in buffer B, r_j is the percentage of streams of type j, and maximum allowed buffer overflow probability to be guaranteed is e^{-a}, then minimum required bandwidth can be calculated by solving (2.5), which requires two optimization procedures.

$$C = \sup_{s}(\inf_{s}(R(s,t))) \tag{2.5}$$

where

$$R(s,t) = \frac{stM\sum_j r_j eb(s,t) + a}{st} - \frac{B}{t} \tag{2.6}$$

In (2.6), $eb(s,t)$ term is found from (2.1).

For a given t, the $R(s,t)$ is a unimodal function of s, having a unique minimizer. Then, $R(s,t) = R_t(s)$ can be solved by using a golden section search method as follows.

1. Given the interval [s_a , s_b], two trial points s_l , s_r are selected such that $s_r - s_a = s_b - s_l = g(s_b - s_a)$, where g is the golden ratio, which is equal to 0.618 roughly.

2. Evaluate $R_t(s_l)$ and $R_t(s_r)$:
   if $R_t(s_l) > R_t(s_r)$ the interval becomes $[s_l, s_b]$
   if $R_t(s_l) < R_t(s_r)$ the interval becomes $[s_a, s_r]$
   if $R_t(s_l) = R_t(s_r)$ the interval becomes $[s_l, s_r]$

3. Steps 1 and 2 are repeated until the uncertainty interval has length less than some small value.

The expectation of $eb(s,t)$ can be approximated by an empirical average. Time parameter t is the most probable busy period of the queue before buffer overflow. Using the measurement data, taken at every epoch time interval τ, t parameter is estimated from (2.5) by calculating $\inf_s(R(s,t))$ values for a range of t values for $t = \tau, 2\tau, 3\tau, , m\tau$ and by choosing the t value for which $\inf_s(R(s,t))$ is maximum. Upper bound for the t candidates, $m\tau$, can be set from buffer limitation. m is bigger for larger buffers. Measurement time window can be used as an upper bound of t. In short, the effective bandwidth optimization procedure is repeated for a range of t values smaller than the measurement window time, and the maximum among them is taken as the required capacity to be allocated.

The run-time of the above procedure depends on the size of the measurement window (i.e. number of epochs of trace window, N), the number of values of t that are processed, and the number of different stream types. For a time parameter of k , empirical calculation of $eb(s,k)$ for any s has $O(N/k)$ computational complexity. The complexity of

the golden section search also scales to $O(N/k)$ when epoch time interval is k. The number of trials in the golden search before finding the optimum s for time parameter k depends on the initial uncertainty interval $[s_a, s_b]$. So overall, the computational complexity scales as $O(N)$. In [2], it is mentioned that algorithm runs satisfactorily in moderate workstations. As for the memory requirement, this method requires holding only N arrival process sample data.

2.2.4 ON-OFF EB Allocation (OOEB)

In the previous algorithms, effective bandwidth is calculated analytically from (2.1). As a second way of using the effective bandwidth concept in resource allocation, we propose a parametric method as this fourth method. The idea is to obtain estimation values of an equivalent On-Off traffic model from measurements, and substitute them in the specific analytical effective bandwidth formula (2.7) for On-Off sources [6].

$$eb(s,t) = \frac{-sr + a + b - 1/2\ (-sr + a - b)^2 - 2\,ba}{2s} \tag{2.7}$$

Parameters a, b and r denote On-Off traffic model where ON and OFF periods are exponentially distributed with parameters a and b respectively, and r is the constant traffic generation rate in the ON state.

The On-Off parameters can be estimated by matching the first three moments of N data measurements falling into the window. But first, this matching is difficult to compute, because moment formulas for On-Off traffic are complex formulas of On-Off parameters. Solution for On-Off parameters can only be done by search algorithms, i.e. computing different values almost arbitrarily to satisfy all three equations at the same time. And second, most of the time, there will not be a match, or fitting On-Off traffic model, since On-Off model has a limited spectrum of traffic types for fitting. There are statistical matching tools, such as [15], where traffic is matched by M state MMPP models. It is shown that M is usually a large number, showing that On-Off model matching is not accurate most of the time.

2.2.5 Norros EB Allocation (NEB)

The Effective Bandwidth formulas mentioned before are based on the assumption that loss probability in a single server queue decreases exponentially as buffer size increases (2.2). However, measurements showed that Internet traffic has self-similar behavior. For self-similar traffic, the decrease in loss probability with increase in the buffer size is slower, due to the self-similar traffic characteristics. This method and the next one are designed with long-range dependent nature of traffic in mind.

In [18], besides introducing modeling of real traffic by fractional Brownian motion (FBM), an effective bandwidth formula for FBM is also given by Norros. FBM model is composed of three parameters, mean rate m, self-similarity parameter H and coefficient of variation a. H and a characterize the *quality* of the traffic in contrast to the long run mean rate m, which characterizes its *quantity* alone.

Queuing analysis of an FBM arrival storage system reveals that the buffer overflow probability decreases hyperbolically with increasing buffer size. The buffer occupancy distribution is approximated by a Weibull distribution. This approximation yields an effective bandwidth formula (2.8).

$$eb = m + \frac{K(H)\sqrt{-2\ln(P_{loss})}}{H} * \frac{a}{2H} * B - \frac{1-H}{H} * \frac{m}{2H} \tag{2.8}$$

where $K(H) = H^H (1-H)^{1-H}$ and m, H, $Ploss$, x and a are mean, Hurst parameter, buffer overflow probability, buffer size and coefficient of variation respectively. The coefficient of variance parameter a, is approximated by the index of dispersion. Although, actually this is a valid assumption only when the traffic is short range dependent.

The Hurst parameter can be set from a priori measurements, for example using long series analysis taking into account diurnal characteristics. However, to react to unexpected traffic changes, a measurement-based on-line algorithm is favored. The H parameter can be estimated from the plot of $log(Var(X(0,t))$ vs. t, but it is shown that this method is unbiased, and needs too many data samples to result in correct estimation. Difficulties of H estimation methods are analyzed in [16]. The comparison of several H estimation algorithms revealed that Abry-Veitch estimator (AV estimator) based on Wavelet theory is shown to be the best way of H estimation [25].

AV estimator takes Discrete Wavelet Transform (DWT), which transforms arrival rate process X into the time-scale wavelet domain. The transform gives observation about

the frequency of data in different time scales, unlike Fourier transform, which gives the frequency of data for the entire time domain. Computation of DWT was not feasible before Multiresolution Analysis Theory appeared and showed that no information is lost if continuous wavelet coefficients are sampled in a dyadic grid in time-frequency plane. As a result, DWT is calculated using a filter bank as in Figure 2.1. The DWT yields a representation of data by means of "details".

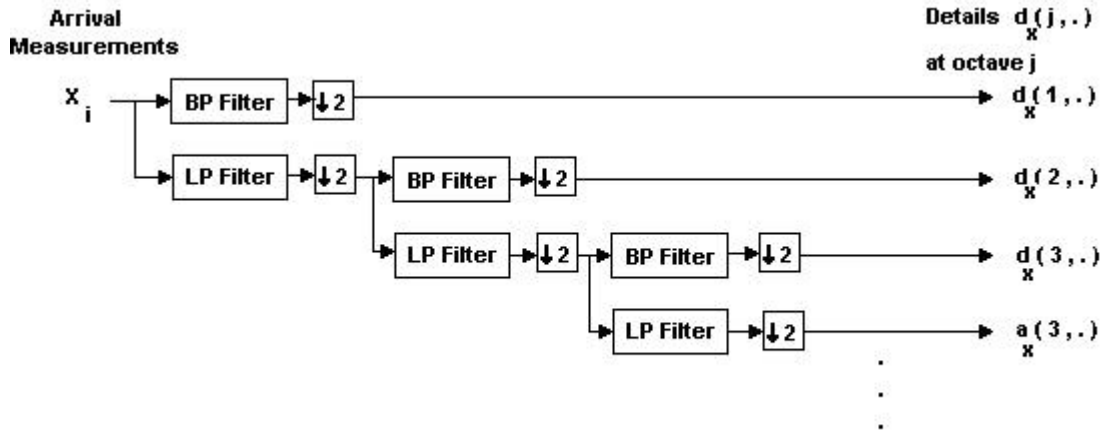

Figure 2.1: Recursive Filter Bank scheme used in DWT calculation.

The detail at octave j is $d_x(j,.)$, where j represents the time scale of observation. If the length of data is N, the number of available octave j is approximately $\log_2(N)$ and the number of detail coefficients approximately halves with each increase of octave due to down-sampling operations in the filter-bank algorithm.

The main feature of the wavelet approach which makes it so effective for the statistical analysis of scaling phenomenon such as LRD is the fact that the wavelet basis functions themselves possess a scaling property, and therefore constitute an optimal coordinate system from which to view such phenomena. The main practical outcome is that the LRD in the time domain representation is reduced to residual short-range correlation in the wavelet time-frequency domain, thus removing entirely the spectral estimation difficulties. Thus for each fixed j, the series $d_x(j,.)$ can be regarded as a stationary process with weak short-range dependence, and these series can be regarded as independent of each other. The

transfer functions of the filters in the filter bank algorithm are formed from continuous time wavelet basis functions. The filters have lag K, and choosing a high value of K protects H estimation from non-stationary effects, such as daily trends.

The complexity analysis of the algorithm relies heavily on the complexity analysis of parameter estimation methods, because once the parameters are estimated, the time of calculation of (2.8) is independent of the number of data samples.

The on-line H estimation algorithm is composed of two parts. First part is the on-line data collection during measurement window of time T. Then, a fast algorithm is run to estimate H when needed, since there is no need to compute H at every data arrival.

When a data sample is available, it is taken as input to the filter bank algorithm. If filters have lag K, (usually $K = 6$ is enough to suppress sufficient non-stationarities.), there passes $(2K+1)$ operations time, which is independent of the number of samples. Due to down-samplings in filter-bank structure, every two data samples will produce a detail sample at octave $j = 1$, every 4 data samples will produce a detail sample at octave $j = 2$, etc. So if N data samples are taken, approximately $N/2$ samples would be collected as details at octave $j = 1$, approximately $N/4$ samples would be collected as details at octave $j = 2$, etc. In a like manner, maximum available octave would be approximately $\log_2(N)$.

Due to the nature of the details, the details at any octave j have zero mean value. In an on-line fashion, variances of details at octave j, $d_x(j,.)$, can be computed as follows. Each time a detail value results from filter-banks, its square is taken and added to the previous sum. So the sum of squares at octave j are calculated. At the end of a time window T, i.e at the final stage of the AV estimator, the estimates of $Var(d_x(j,.))$ is obtained by dividing the sum by number of detail values at octave j. The rest is to estimate H from the variation of $Var(d_x(j,.))$ vs j, which can be done by linear regression fitting in the linear part of the plot. The number of available octaves j is $\log_2(N)$, and at the end of the windows, all required is a linear fit to the $\log_2(N)$ numbers, (which is a linear operation). Consequently, the computational complexity of the algorithm scales to $O(\log_2(N))$.

AV estimator has small memory requirement, since maximum octave j is $\log_2(N)$, keeping sum of squares and number of values available at each octave results in a memory requirement of $2\log_2(N)$ numbers. And also note that variance estimates at octaves close to $\log_2(N)$ comes from a small number of $Var((d_x(j,.))$ values (due to down-samplings). Therefore they are not reliable and should be discarded. Typically, if data arrivals are

sampled in 10ms. intervals, and a window of 10s. is used for the time between H estimations, there would be 1000 data samples. About 9 octaves would be available, with about 500 samples of $d_x(j,.)^2$ at octave 1, about 250 samples at octave 2, etc.

2.2.6 DRDMW (Improved Empirical EB Allocation)

This method [23] is an improved version of DEB method in two ways. First, a unified phenomenological framework to estimate overflow probability of both long range dependence (LRD) and short-range dependence (SRD) is put forward by including the Hurst parameter in traditional analytical effective bandwidth methods, as in (2.9).

$$P(B < Q) = e^{-s(C)B^{2-2\,H}} \tag{2.9}$$

Second, the difficulty of measuring the effective bandwidth of real-time traffic online by using direct estimator [23] is alleviated by using an approach based on dual recursive algorithm with double moving windows (DRDMW), which is introduced in empirical calculation of analytical effective bandwidth formula instead of using direct estimator.

This algorithm requires the mean source rate to calculate the sliding window interval in the measurement window. The algorithm uses this a priori traffic information to reduce overflow risk by decreasing the frequency of the analytical effective bandwidth formula recalculation.

Similar to DEB, DRDMW has a computational complexity of $O(N)$, assuming H is estimated in real-time as described in NEB.

2.2.7 Gaussian Approximation Allocation (GA)

All the algorithms mentioned so far have implementation difficulties. We test their computational complexities with respect to the simplest resource allocation method existing in literature, Gaussian approximation method [9]. Although this method is expected to have the poorest performance, our aim is to investigate whether or not the loss of performance would be acceptable compared to computational simplicity, comparing to other proposed algorithms.

Incoming traffic arrival rate distribution can be assumed to be Gaussian if there is a sufficient degree of aggregation of sources. In Gaussian Approximation, buffer is ignored

and server capacity is set according to gaussian arrival rate distribution as in (2.10) so that the loss probability requirement is satisfied, since tail probability (probability that arrival rate is greater than server rate) gives the desired loss probability.

$$C = m + \sigma * \sqrt{-2 * \ln(P_{loss}) - \ln(2 * \pi)} \tag{2.10}$$

where m , σ are the mean and standard deviation of the arrival rate distribution.

2.3 Comparison of the Algorithms

The first algorithm, namely Direct Effective Bandwidth Allocation algorithm, relies on the effective bandwidth formula, and possesses the problem of finding appropriate values for s and t, which depend on QoS requirements and the system parameters. The space parameter is estimated using the Large Buffer Assumption. The time parameter estimation is left somewhat arbitrary, for the time being.

The second algorithm uses (2.3), which is an alternative generic effective bandwidth definition in terms of the mean rate, index of dispersion, QoS parameter and buffer size. It is simpler, but it still does not address long range dependent traffic.

The Many Sources Asymptotic Effective Bandwidth algorithm relies on the effective bandwidth formula (2.1) and encounters the problem of estimation of (s, t). This method accomplishes it by solving a functional optimization problem. Although it is a very innovative approach, this may be too slow for our motivational *self-sizing* scenario where every node takes on-line measurements of every traffic type.

The On-Off Effective Bandwidth formula (2.7) for our fourth method is obtained by substituting an On-Off arrival process instead of $X(0, t)$ in the analytical effective formula. With regard to a practical usage of such expressions, we encountered other problems than estimation of (s, t) parameters, such as model parameter estimation, and goodness of fit of the model.

The Norros Effective Bandwidth Allocation and Gaussian Approximation methods are alternatives which do not include non-trivial (s, t) parameter estimations. They are approximate expressions, which are derived independently of the effective bandwidth formula. The Gaussian Approximation algorithm assumes a bufferless link. This will overestimate required capacity. Moreover, the gaussian assumption is not valid for traffic formed by small

number of sources. This places a constraint on the source type, however our aim is to have an algorithm capable of functioning without unreasonable assumptions.

The self-similarity is addressed only in the NEB and in the DRDMW method. Others do not discriminate between short range dependence and long range dependence. Although the index of dispersion in the Courcoubetis formula of the second algorithm stands for burstiness of the source, the formula is not for long range dependent traffic. Thus the effective bandwidth approximation on which Courcoubetis formula is based, (i.e., exponential decay of buffer overflow probability with increasing buffer size) is not valid for self-similar traffic (the decay is hyperbolic and slower than exponential). We provide a summary of our performance comparisons with respect to various network scenarios in Table 2.1.

Table 2.1: Performance Comparisons

	MSAEB	**DEB**	**OOEB**	**CEB**	**NEB**	**DRDMW**	**GA**
Small Buffer	Very Good	Poor	Poor	Poor	Good	Poor	Perfect
Large Buffer	Very Good	Very Good	Very Good	Very Good	Very Good	Very Good	Poor
SRD Traffic	Good	Very Good	Good	Very Good	Good	Very Good	Good
LRD Traffic	Good	Poor	Poor	Poor	Perfect	Poor	Poor
Many Sources	Perfect	Very Good	Poor	Very Good	Very Good	Very Good	Perfect
Single Source	Poor	Very Good	Good	Very Good	Good	Very Good	Poor
Computational Complexity	Poor	Poor	Poor	Very Good	Good	Good	Perfect
Memory Requirement	Good	Good	Perfect	Good	Good	Perfect	Perfect

The Gaussian Approximation algorithm is the easiest to implement, and suitable to be used as an algorithm setting an *upper bound*, since it does not consider buffer size. The Courcoubetis Effective Bandwidth Allocation is also another easy, and promising one, since this one takes into account buffer also. But neither of the previous two algorithms is designed with long range dependent traffic in mind. Norros effective bandwidth and DRDWM algorithms are the only ones incorporating the Hurst parameter, therefore ad-

dressing to long range dependent traffic. Although DRDMW is designed to alleviate the numerical overflows in the direct effective bandwidth allocation, that problem can not be completely alleviated due to the structure of (2.1). Therefore, we selected the following three algorithms for further simulation analysis:

- Gaussian Approximation (GA)
- Courcoubetis Effective Bandwidth Allocation (CEB)
- Norros Effective Bandwidth Allocation (NEB)

2.4 Summary

In this chapter, the results of our literature survey seeking for suitable methods for our on-line capacity allocation purpose are presented. The analytical predicted performance comparisons are summarized in Table 2.1. In the following chapter, three selected algorithms are analyzed through further simulation study.

Chapter 3

Comparison of Methodologies via Simulation

3.1 Introduction

In this chapter, we present the simulation results of the algorithms selected in the previous chapter. First, we explain our simulation methodology. Then, the results are given in section 3.3 under 3 subsections. Section 3.3.1 exhibits performance results. Section 3.3.2 shows how much price is paid while obtaining these results. Finally, section 3.3.3 provides the computation complexities of the algorithms as a function of the measurement window size.

3.2 Simulation Methodology

The selected on-line measurement-based resource allocation algorithms are implemented and tested in a simulation scenario as shown in Figure 3.1. This is a single server queue simulation where the service rate is changed, in an on-line fashion, periodically based on recent traffic measurements. The simulations are replicated with different values of the measurement slot and window sizes.

We used the Sup-FRP traffic model [22], to generate the incoming traffic. The simulation flow slides packet by packet, emulating a real case scenario as in an Ethernet card passing packets to upper network layers.

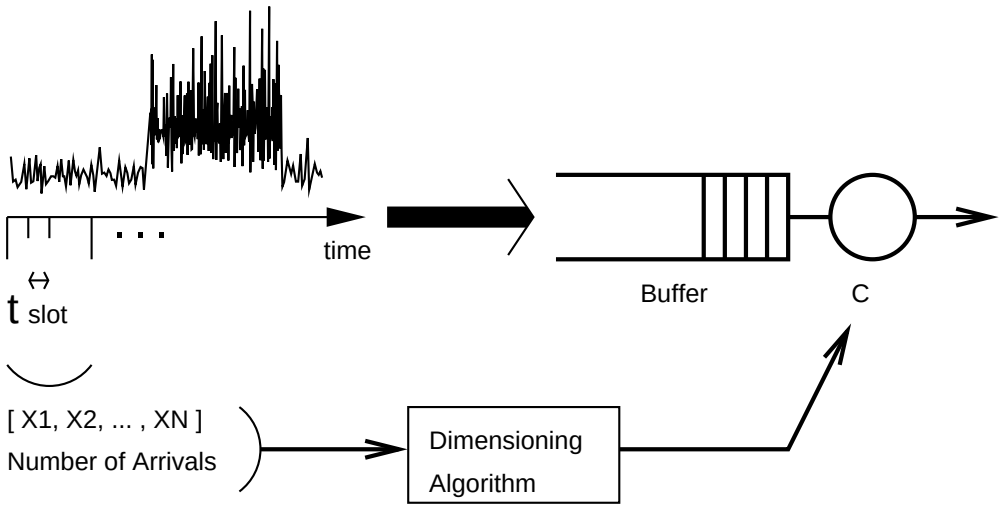

Figure 3.1: Simulation scenario.

Figure 3.2 gives a visual representation of how algorithms adjust service rates, tracking fluctuations in the incoming traffic rates, so as not to waste resources.

In our simulations, the performance metric is related to a QoS constraint, in our case packet loss probability. To quantify the amount of expanded resources, we use two cost metrics:

- **Allocation Ratio** (Average Capacity Allocation divided by Average Traffic Rate)
- **Average Queue Occupancy**

We performed simulations with 5 different t_{slot} values (0.01, 0.05, 0.1, 0.5, 1s) and 5 window size (N) values (3, 6, 30, 60, 300slots) in every method. Therefore, we had 25 simulations per method. As a total, we present here results of 75 simulations. Also note that the measured statistics resulted after 30 simulation replications and confidence intervals are insignificant.

In all of the simulations in this section, we generated traffic with the same mean value of 20 Kbytes/s, the same Hurst parameter of 0.7 and the same buffer size of 5 Kbytes.

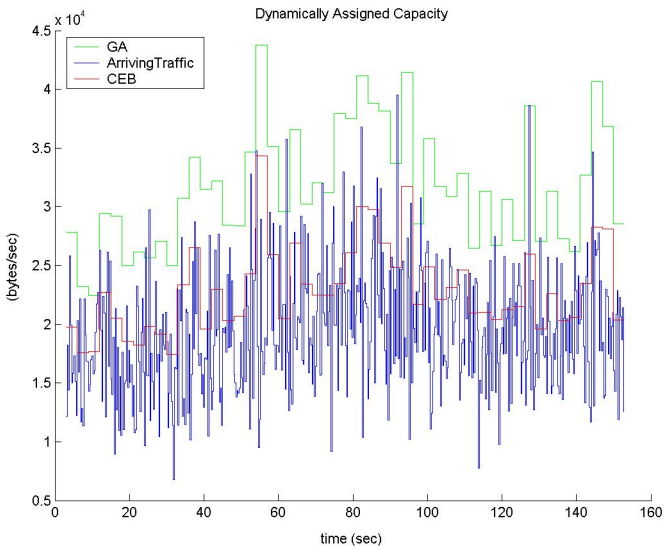

Figure 3.2: A visual example view of dynamic capacity allocation.

We set the QoS target to packet loss probability of 10^{-3}, so as to have a common ground for the performance comparisons of algorithms in the simulation scenario (Figure 3.1).

Note that the average traffic rate of 20 Kbytes/s is the product of an average packet size of 200 bytes and average packet arrival rate of 100 packets/s. Therefore, the average time between two consecutive packets is 0.01 seconds and the t_{slot} values chosen in the simulations, which are (0.01, 0.05, 0.1, 0.5, 1 s), correspond to cases where 1, 5, 10, 50 and 100 packet arrivals take place on average in a slot time duration, respectively. Also note that the choices for t_{slot} and N are made deliberately to have simulations where reallocation takes place in every 3 seconds, but with different measurement resolution in the recent history of the measurement data. For example, a simulation with (0.01 s, 300 slots) includes 300 measurements, whereas the one with (1 s, 3 slots) includes three measurements in the same recent 3 seconds history.

We did not implement an on-line H estimation [21]. We provided the value of H (i.e., 0.7) to the algorithms beforehand, so that we can examine the performance of the bandwidth allocator, independent from the performance of the H estimator. The combined, on-line Hurst parameter and EB estimation is beyond the scope of this thesis and is left as future work.

3.3 Simulation Results

In this section, we present performance analysis and demonstrate the importance of the time scale choice in measurement-based algorithms.

The plots in this section are in the form of three dimensional graphs. The xy plane is composed of slot duration and window size pair, i.e., (t_{slot}, N). There are 25 z coordinate points coming from the convolution of 5 values of t_{slot} and 5 values of N, which are (0.01, 0.05, 0.1, 0.5, 1 s) and (3, 6, 30, 60, 300 slots), respectively. We chose to present simulation results in this form, rather than listing the numbers in a table form, because it would be difficult in the table form to observe the trends of the data when the time slot length increases while the window size is kept constant and vice a versa. At the end of this section, we provide the processing time plots with respect to the window sizes.

3.3.1 Performance Plots

The following three plots show the performance results for different t_{slot} and N values in the simulation scenario given in Figure 3.1. Figure 3.3 shows the loss probability results when GA is used for link dimensioning.

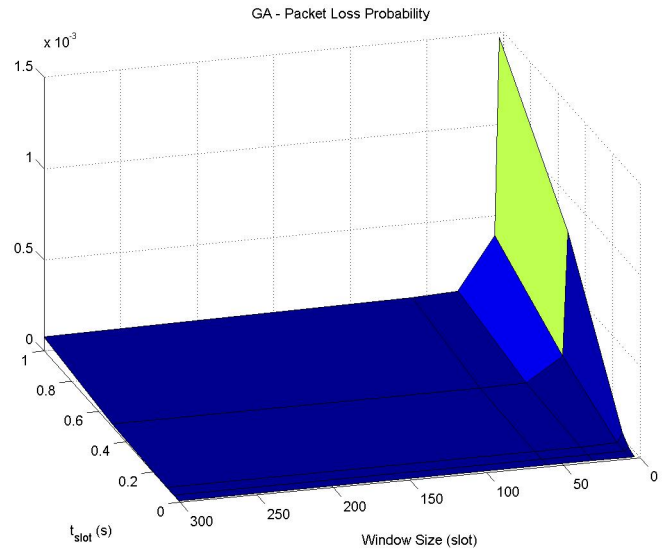

Figure 3.3: Loss probability against different time slot length and window size values, when GA is used for dynamic resource allocation.

As seen from Figure 3.3, the loss probability increases when t_{slot} is increased while N is kept constant. Furthermore, it can be deduced that the loss probability decreases when N is increased while t_{slot} is kept constant.

The QoS target, which is 10^{-3}, is satisfied in every (t_{slot}, N) combination with the exception of (t_{slot} = 1 s, N = 3 slots). Note that GA is used as an upper band of resource allocation for comparison purposes. It does not consider buffer size. In fact, it assumes there is no buffer. This is why, it is expected to be more generous than other algorithms. The fact that a violation of QoS took place in this method implies trouble for other methods.

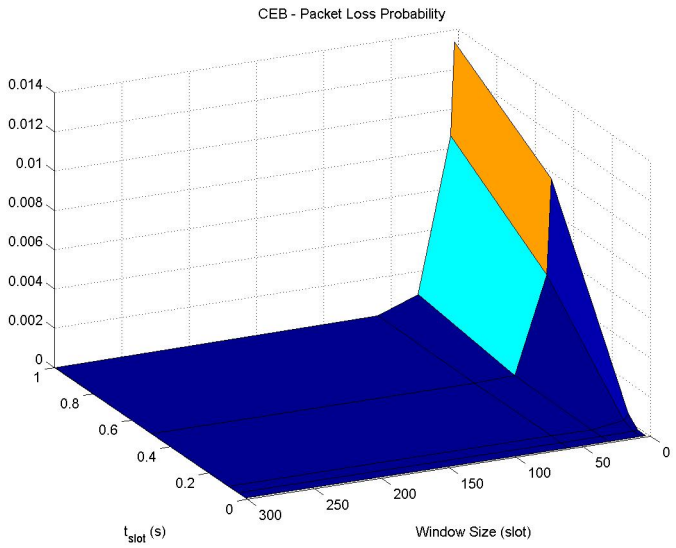

Figure 3.4: Loss probability against different time slot length and window size values, when CEB is used for dynamic resource allocation.

Figure 3.4 tells us that the CEB's performance changes similar to GA against t_{slot} and N variations, but P_{loss} values are relatively about an order of magnitude *higher*. The QoS target is violated for the following (t_{slot}, N) pairs: (0.5 s, 3 slots), (1 s, 3 slots), (0.5 s, 6 slots) and (1 s, 6 slots). This algorithm, considering the presence of buffer, theoretically permits lesser resource usage than GA.

The loss probability results of NEB are provided in Figure 3.5. For N values of 3 and 6, the P_{loss} values are between the ones of GA and CEB. However, when N is either 30, 60 or 300, P_{loss} is smaller than other algorithms, which implies an over-allocation of

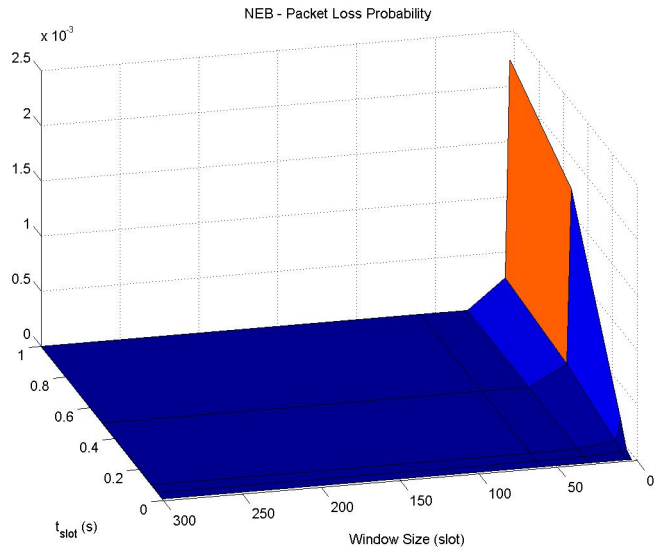

Figure 3.5: Loss probability against different time slot length and window size values, when NEB is used for dynamic resource allocation.

bandwidth.

3.3.2 Cost Plots

The plots in this section show how much resources are used while obtaining the performance results introduced in the previous section.

The following two plots are cost plots for GA. Figure 3.6 agrees with Figure 3.3 and shows that as t_{slot} is increased for a constant N, the allocation ratio decreases towards 1. We also observe that for constant t_{slot}, increasing N results in a larger capacity allocation. But this rate of increase in capacity allocation depends on the t_{slot} value. For instance, when t_{slot} length is fixed to 1s, the allocation ratios for N =3, 6, 30, 60 and 300 are 1.34, 1.40, 1.46, 1.47 and 1.51, respectively. However, when t_{slot} is set to 0.01 s, the corresponding ratio values are 4.21, 5.02, 5.89, 6.23 and 9.28. So, once t_{slot} is *properly* chosen, choosing N looses its importance, since the change in the ratio values are much smaller.

Figure 3.7 is in accordance with Figure 3.6 and Figure 3.3, and gives an idea about how much buffer is occupied. Figures 3.3, 3.7 and 3.6 show the importance of time scale choice. We observe that even if GA does not consider buffer, it can still be an acceptably

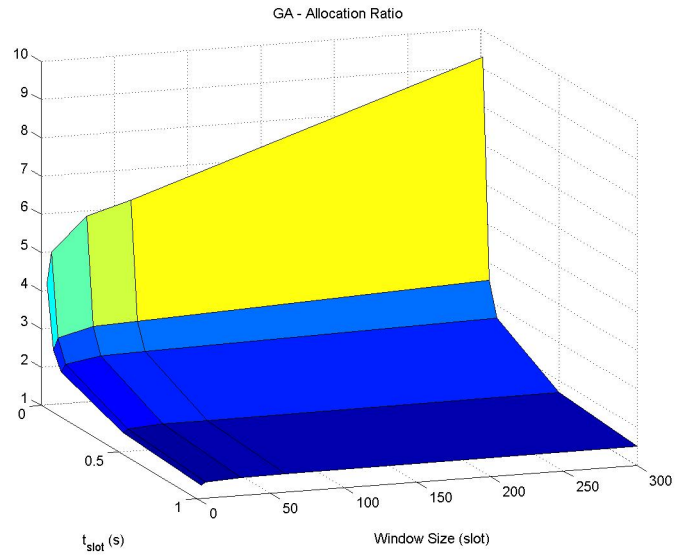

Figure 3.6: Ratio of the average capacity allocation to the average traffic rate against different time slot length and window size values, when GA is used for dynamic resource allocation.

good resource allocator, given that t_{slot} and N values are chosen properly.

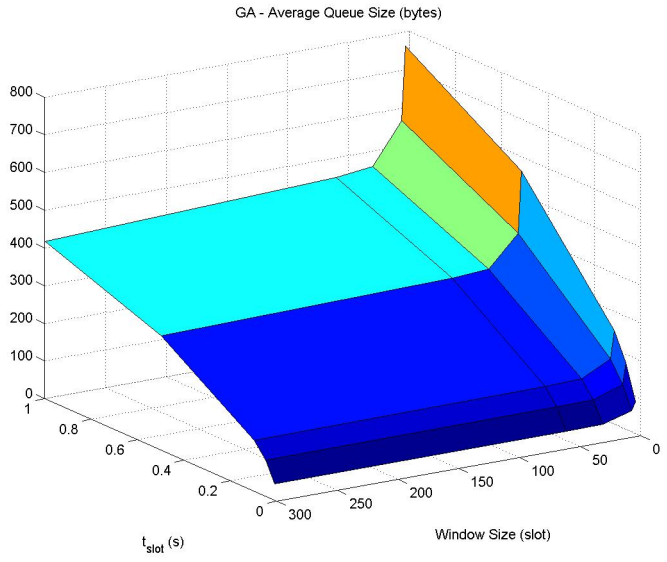

Figure 3.7: Average Queue Occupancy against different time slot length and window size values, when GA is used for dynamic resource allocation algorithm.

Figures 3.8 and 3.9 are cost plots when CEB is used as the dynamic resource allocation method. Figure 3.9 shows that the buffer occupancy changes drastically with the choice of N. For small N, the occupancy is bigger than the one for GA, but for big window size values (60 and 300), the buffer is less occupied than when GA is used as resource allocator. These implications are justified in Figure 3.8. As a result, this shows that if N is chosen poorly, the resource allocation can be even worse than GA's allocation.

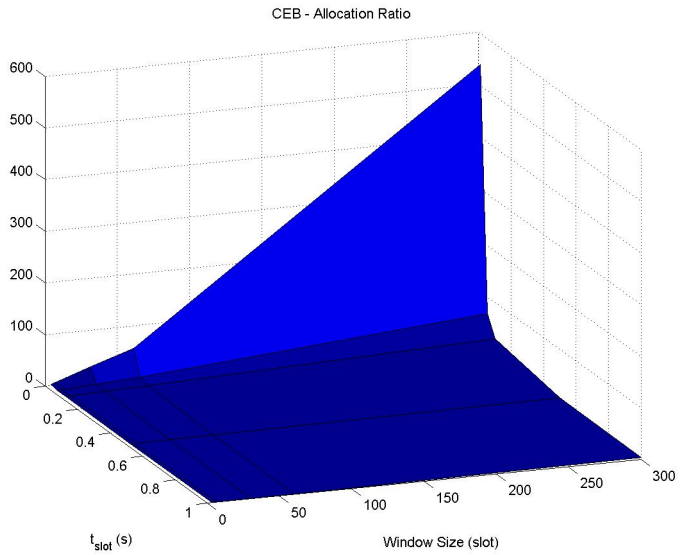

Figure 3.8: Ratio of the average capacity allocation to the average traffic rate against different time slot length and window size values, when CEB is used for dynamic resource allocation.

Compared to GA, we observe that bigger t_{slot} values lead to better allocation ratios (i.e., ratios closer to 1) and the choice of N has a greater effect on CEB. With proper choice of t_{slot} and N, the same performance can be achieved with lesser resource usage. To illustrate, when t_{slot} is fixed to 1s, the allocation ratios for N 3, 6, 30, 60 and 300 are 1.04, 1.08, 1.29, 1.54 and 3.88 respectively, and when t_{slot} is set to 0.01s, the corresponding ratio values are 4.86, 8.45, 30.66, 58.25 and 540.13. Here, similar to GA, we see the importance of choosing t_{slot} properly. But unlike for GA, here choosing N is also important. This is because the rate of increase of ratio values when N is increased is much more significant. Choosing a large N leads to serious over-allocation. Also note that in the above cases where allocation ratios are 1.04 and 1.08, the P_{loss} values are 0.0134 and 0.0087 respectively, and

the QoS criterion of 10^{-3} is not satisfied by one order of magnitude.

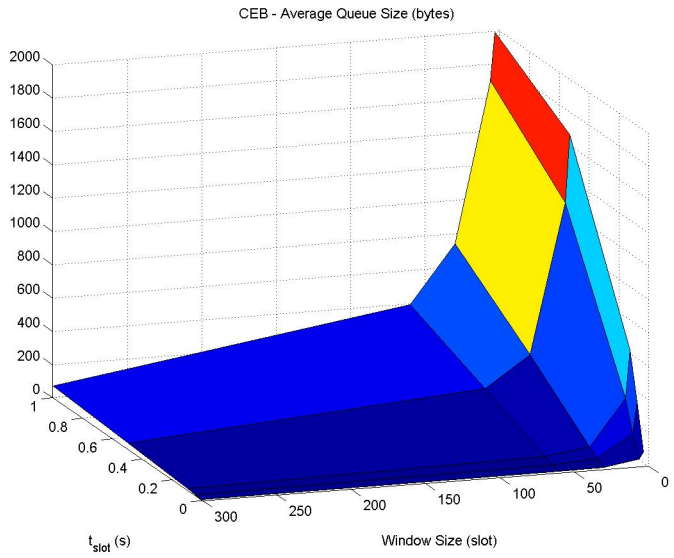

Figure 3.9: Average Queue Occupancy against different time slot length and window size values, when the CEB is used for dynamic resource allocation.

Figures 3.10 and 3.11 are cost metrics plots when NEB is used as the dynamic resource allocation method. Figure 3.10 shows that there exists over-allocation of resources when the window size is 30, 60 and 300, similar to the behavior observed in CEB.

When it comes to the choice of t_{slot}, as in the previous methods, a bigger t_{slot} resulted in better resource allocation. This method is the only one which allocates more capacity to the traffic possessing higher long-range dependence. Overall, it can be said that NEB includes similar performance and cost changes as the ones of CEB, but performance values are around one order of magnitude better, and consequently, cost values are higher. For example, for t_{slot} and N pairs of (1 s, 3 slots) and (1 s, 6 slots), the P_{loss} and allocation ratio values were 0.0134, 0.0087 and 1.04, 1.08, respectively, for CEB, but the corresponding values for NEB are 0.0022, 0.0003 and 1.38, 1.61 respectively. This shows that NEB has a tendency of allocating more resources than CEB, and results in better QoS constraint satisfaction.

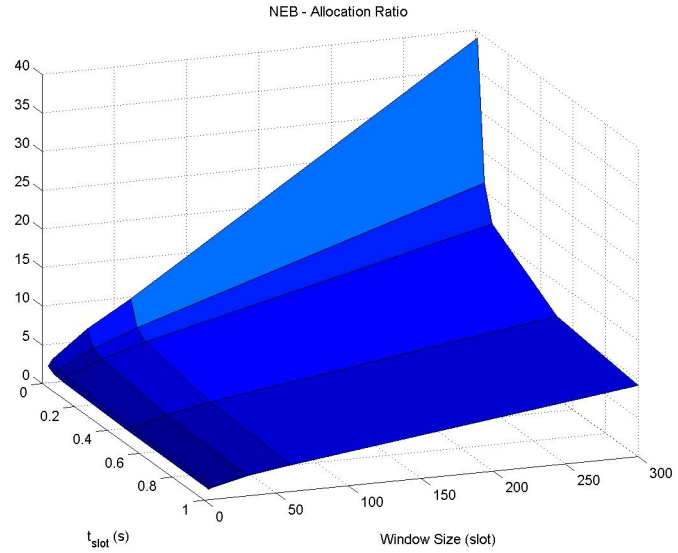

Figure 3.10: Ratio of the average capacity allocation to the average traffic rate against different time slot length and window size values, when NEB is used for dynamic resource allocation.

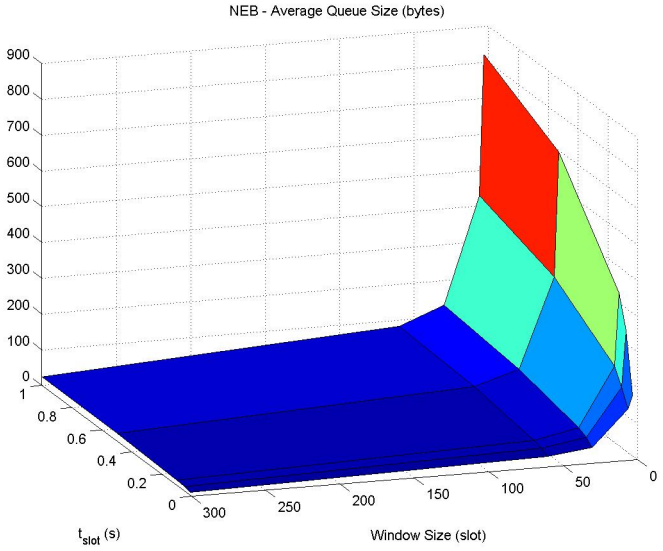

Figure 3.11: Average Queue Occupancy against different time slot length and window size values, when NEB is used for dynamic resource allocation.

3.3.3 Complexity

Figure 3.12 shows the processing times of the algorithms as a function of the window size. The processing time is the time required for the algorithm to re-calculate capacity. The processing time is seen to be related to N for CEB and NEB, with a complexity of $O(N^2)$. This result is in parallel with our expectations. CEB and NEB calculate autocorrelations of measurements falling into the measurement window of size N, and this requires a processing time proportional to N^2. Whereas, GA uses only the mean and variance of the measurements, whose calculations are fully on-line. As a result, GA has complexity of $O(1)$ and regardless of N, its execution time remains much smaller compared to other algorithms.

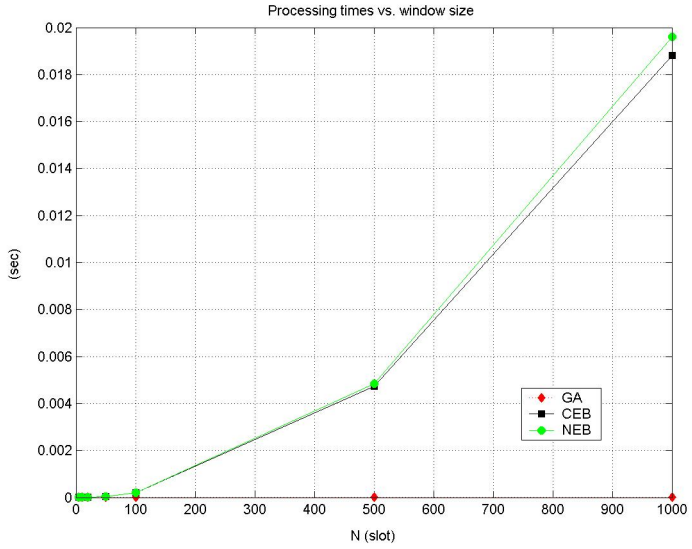

Figure 3.12: Processing times of algorithms vs. window size.

3.4 Summary

In this chapter, the simulation methodology and results are given. The algorithms performed as expected after the analytical analysis. The plots indicate the importance of the measurement time scale and the measurement window size, which is investigated further

in detail in chapter 5. The results of this chapter encouraged us to perform the practical implementation given in the next chapter.

Chapter 4

On-line Bandwidth Estimation and Link Sizing Experiment

This chapter shows the applicability of the on-line measurement based capacity allocation algorithms in a practical scenario.

4.1 Overview of the Experiment

The network emulation environment is shown in Figure 4.1.

We implement our on-line capacity allocation algorithms (i.e., Gaussian Approximation Effective Bandwidth (GA), Courcoubetis Effective Bandwidth (CEB) and Norros Effective Bandwidth (NEB)) into a computer running Linux operating system (the *ingress* in Figure 4.1). We emulate a network environment where *ingress* is an edge computer of a *core1* network, which accepts connections from *carolina*, *ncstate* and *wolfpack*. Each of these latter three emulate a "network" communicating with *core1* "network" whose edge router is *ingress*. The node *ingress* reads and classifies packets incoming to the network *core1* from these three "networks". At the same time, *ingress* also measures the traffics by means of the NeTraMet software (Network Traffic Measurement Architecture, described by RFC's 2720-2724).

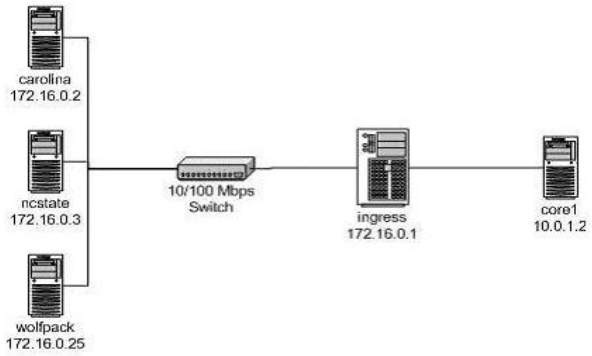

Figure 4.1: Experiment network diagram.

The logical experimental setup diagram is as follows.

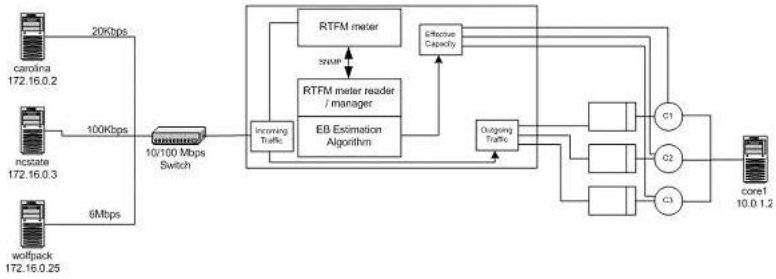

Figure 4.2: Logical diagram of the experimental setup.

NeTraMet architecture has a traffic meter and traffic reader which can communicate through SNMP. Therefore a meter and a reader can be in separate computers in different networks. But note that we have them at the same computer, namely *ingress*. NeTraMet is designed to be a standard as a network measurement architecture. We use NeTraMet in the classification and measurement of streams coming from the *carolina*, *ncstate* and *wolfpack* "networks". Basically, NeTraMet gives the total number and the source "network" of the incoming packets every t_{slot} interval. Having the window size N, N of such information is used in the estimation of the effective bandwidths for each stream. Finally, *ingress* uses these effective bandwidth estimations to change the bandwidths allocated in the *core1* link. This is performed by means of Linux Traffic Control utility, TC [11]. This allocation takes place every $t_{slot} * N$ time interval. We use $t_{slot} = 1$s and $N = 10$.

In short, *ingress* not only estimates three effective capacities based on measurements in an on-line fashion, but it also changes the actual capacities allocated to each stream between *ingress* and *core1* every 10s. We also measure the traffic incoming to *core1*. By comparing the sequence numbers, we measure the actual loss probability, and compare it with the target loss probability value used in our effective bandwidth estimation algorithms, which is 0.001.

4.2 Experiment Results

We present capacity allocation and packet loss probability plot pairs of three experiments, in which different effective bandwidth estimation algorithms are tested (namely GA, CEB and NEB). The capacity allocation plots shows the total traffic incoming to *ingress* and the capacity allocations of the three streams based on the effective bandwidth estimations. Total capacity allocation is also plotted, so that a pictorial comparison of the total bandwidth allocation vs. total incoming traffic can be made. Different than capacity plots, the packet loss probability plots are obtained from the comparison of the original traffic data with the measurements taken in *core1*.

Note that the effective bandwidth curves in Figures 4.3, 4.5 and 4.7 are shorter than the traffic length (which is 1000 seconds). We appended zeros at their ends. The cause for this was that when an enquiry is made from NeTraMet software, it takes on average 1.08225 seconds to return the measurements along with mean/variance/capacity

calculations. Having called this process every slot, i.e. 1000 times, means 82.225 seconds are lost due to computing time, giving a value of around 920 measurement points instead of 1000. One can increase the t_{slot} time, this will decrease but not prevent the amount of loss packets. In a real application such as this one, this was a difficulty not encountered in our simulation studies. However, this problem can be alleviated mostly by bypassing the NeTraMet software, and writing a network sniffer software, specialized in measurements. As mentioned previously, NeTraMet is designed with different considerations and it includes unnecessary parts for our purpose. For instance, SNMP causes much of the overhead in the meter reading, which is not needed in our case, since we have the meter and reader at the same computer.

Figure 4.3 gives the capacity allocation of individual streams, as well as the total allocated capacity (i.e. sum of allocated capacities for all of the streams between the ingress and core1 node), when the link bandwidth is dynamically adjusted using the GA algorithm. We observe that the total capacity allocation is between the mean and peak of total incoming traffic rate. The trajectory of the packet loss probability measurements of the 3 streams in the core1 node is given in Figure 4.4.

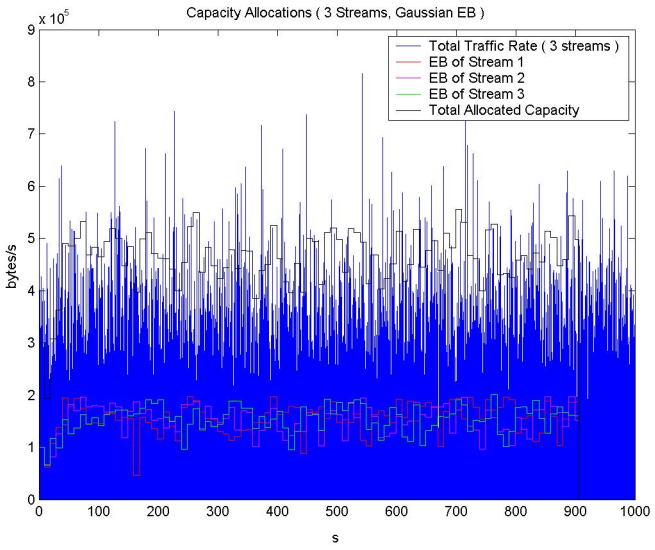

Figure 4.3: A view of total traffic rate and individual capacity allocations of the 3 streams in the Ingress node, where Gaussian EB is used in capacity estimation.

Figures 4.5, 4.6 and 4.7, 4.8 are similar to Figures 4.3, 4.4 respectively, but different

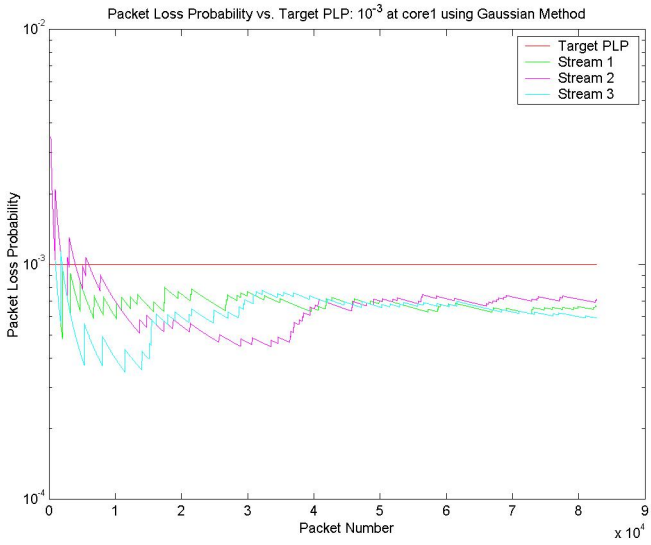

Figure 4.4: Loss Probability of 3 streams measured at *core1* (when Gaussian EB is used as capacity allocator in *ingress*).

capacity allocators are used in *ingress*. The CEB algorithm case is given in Figures 4.5 and 4.6. Whereas Figures 4.7 and 4.8 are for the NEB algorithm case.

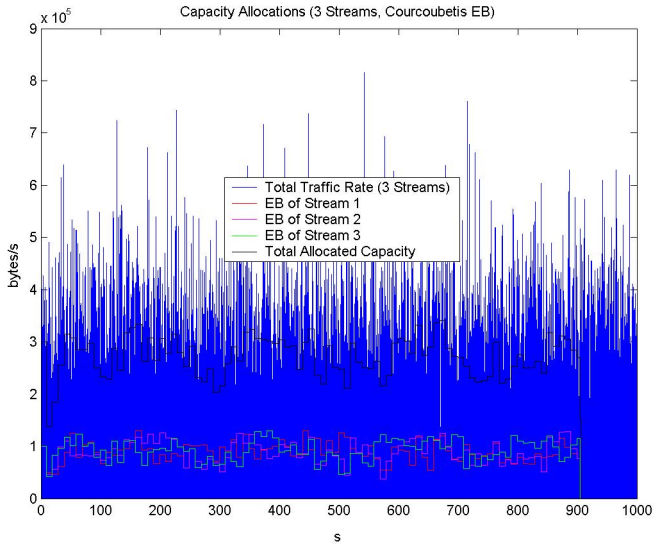

Figure 4.5: A view of total traffic rate and individual capacity allocations of the 3 streams in the Ingress node, where Courcoubetis EB is used in capacity estimation.

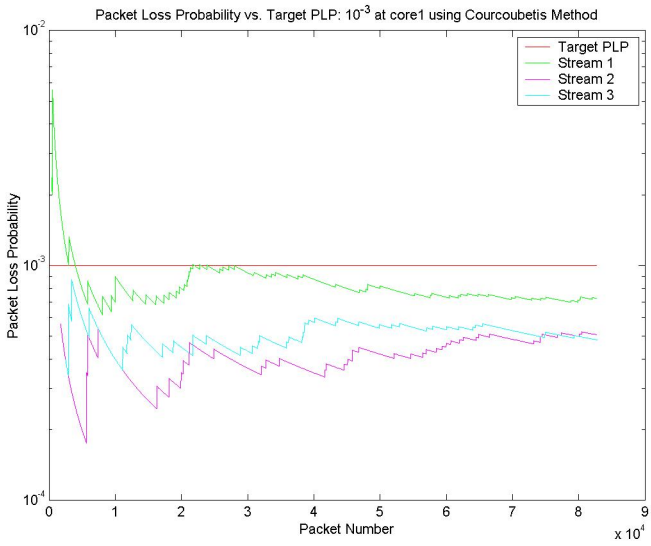

Figure 4.6: Loss Probability of 3 streams measured at *core1* (when Courcoubetis EB is used as capacity allocator in *ingress*).

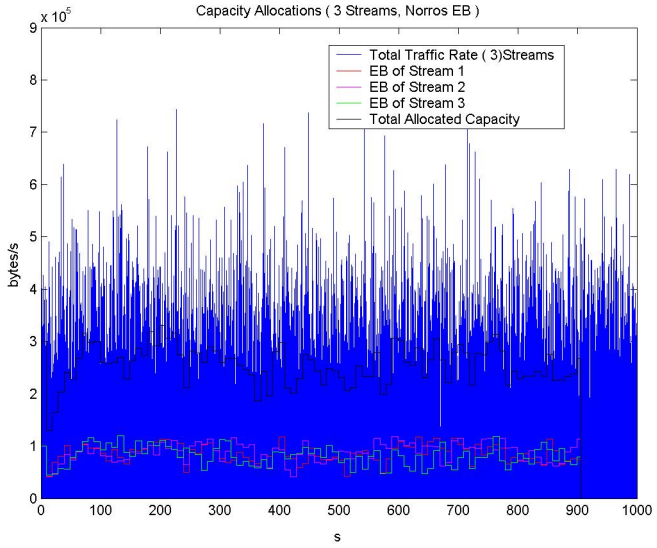

Figure 4.7: A view of total traffic rate and individual capacity allocations of the 3 streams in the Ingress node, where Norros EB is used in capacity estimation.

From the figures, we observe that the total capacity allocation is between the mean and peak of the total incoming traffic rate for all of the algorithms. Table 4.1 gives

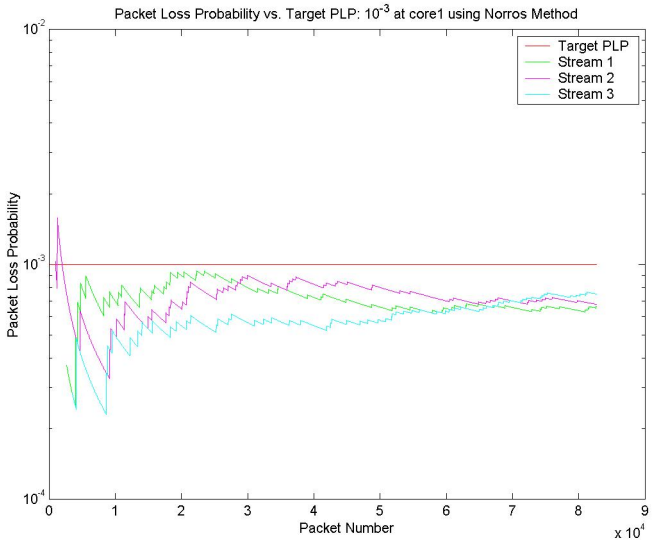

Figure 4.8: Loss Probability of 3 streams measured at *core1* (when Norros EB is used as capacity allocator in *ingress*).

the percentages of the saved bandwidth compared to a Maximum (Peak) Rate Allocation algorithm. As for the cost paid in obtaining this performance improvement, we observe that the loss probability trajectory plots are kept below the target loss probability value of 0.001. These values agree with simulation results in chapter 3 when t_{slot} is 1s and N is 10slots.

Table 4.1: Saved Bandwidth Compared to a Maximum Rate Allocation Algorithm

	Gaussian EB	Norros EB	Courcoubetis EB
Percentage Decrease in the Bandwidth Usage Compared to Maximum Rate Allocation	%43.10	%67.14	%68.45

4.3 Summary

We implemented our on-line bandwidth allocation algorithms in a real network node where link sizing is performed based on the measurements. Algorithms saved substantial amount of bandwidth compared to a Maximum Rate Allocation algorithm, while still satisfying the QoS constraint. Although we faced a practical implementation problem, we obtained successful results in general and this real implementation inspired us for further practical use of our effective bandwidth algorithms. Actually, more important than the practical problem we faced, we believe the main obstacle in front of porting this implementation into a real implementation is the "constant" time scale choices. The experiments are performed with specific t_{slot} and N values of 1s and 10slots. In the next chapter, we explain the disadvantage of using static time scale, and propose an algorithm to resolve this problem.

Chapter 5

Dynamic Time Scale

5.1 Introduction

Chapter 3 emphasized the importance of the measurement time scale choice. In this chapter, we introduce the dynamic measurement time scale approach.

This chapter is organized in the following outline. Section 5.2 interprets the response of the algorithms to the time scale changes, in the light of chapter 3, where we observe that a satisfactorily performing algorithm can experience a performance degradation when only the measurement time scale is changed. Then, we reason that the measurement time scale is dependent on the input traffic. That is, we claim that an algorithm working good with a measurement time scale for an input traffic can work poorly after changing only the input traffic. This observation yields us to deduce that the measurement time scale is *relative* to the input traffic. Accordingly, we propose to change the measurement time scale dynamically, based on measurements themselves. Section 5.3 defines the dynamic time scale. Finally, simulation results showing the robustness of this approach over the static time scale methods are presented in section 5.4.

5.2 Relevance of Dynamic Time Scale

Sections 3.3.1 and 3.3.2 show how drastically the performances of the measurement-based capacity allocation algorithms change depending on the time scale choice, as pointed out in [8].

Mainly, we observed that increasing the measurement slot, t_{slot}, results in a decrease in the capacity allocation and consequently an increase in P_{loss} in all of the algorithms. This can be explained intuitively from the structure of the formulas used in the capacity allocation algorithms. To illustrate, consider the Gaussian Approximation Algorithm's formula (2.10), where m and σ are the mean and standard deviation of the most recent N measurements, $[X_1, X_2, ..., X_N]$. Each X_i represents incoming traffic load in consecutive t_{slot} durations. By the law of large numbers, as t_{slot} increases, say $t_{slot} \rightarrow A$, where A is a time parameter, which is *large* (dependent on the traffic characteristic), then X_i approaches $m * A$ for all i. This causes σ in (2.10) to go to zero, and the capacity value, eb to approach to the mean rate, m. A similar reasoning can be given for other methods, in which not only standard deviation, but also autocorrelations of measurements, $[X_1, X_2, ..., X_N]$ are used.

On the other hand, we also observe that taking t_{slot} arbitrarily *small* ends up in over-allocation of resources. As t_{slot} decreases, the measurement history ($t_{slot} * N$) decreases too. This decreases the confidence and increases the randomness in the formula parameter estimations, yielding over-allocations.

We could obtain empirical t_{slot} values from the performance and cost metrics plots, so that the QoS is satisfied with minimum resource allocation. However this particular t_{slot} value would be useful only for the traffic that we used in our simulations. Consider using a traffic whose mean is $m * K$ (that is K times bigger). This time, on average K times more traffic load will fall on average into the slots. Relatively, this is the same experiment as using K times bigger t_{slot} measurement slots, with mean traffic rate m. In other words, the measurement time scale is *relative* to the traffic characteristics. A static t_{slot} may correspond to cases where we described previously as *small* or *large*, depending on the incoming traffic.

As a result, we believe that the measurement time scale t_{slot} should also change dynamically based on measurements $[X_1, X_2, ..., X_N]$, in order to keep the algorithms always working close to their best.

5.3 Definition of Dynamic Time Scale

In [5], the Maximum Time-Scale (MaxTS = t^*) is used as the time scale of interest for queueing systems fed by a fractal Brownian motion (fBm) process:

$$t^* = \frac{k\sigma H}{(C-m)}^{\frac{1}{1-H}} \tag{5.1}$$

where $k = \sqrt{-2 * \ln(P_{loss})}$, m is the mean traffic rate, σ is the standard deviation of the traffic rate and C is the capacity of the server.

The value of t^* is derived from (5.2), where $\hat{A}_H(t)$ is the probabilistic envelope process of the fBm cumulative arrival process $A_H(t)$ ($A_H(0) = 0$), such that $P(A_H(t) > \hat{A}_H(t)) \approx P_{loss}$:

$$\frac{d\hat{A}_H(t^*)}{dt} = C \tag{5.2}$$

The fBm envelope process is defined in [5] as

$$\hat{A}_H(t) = mt + k\sigma t^H \tag{5.3}$$

On the basis of the law of large numbers, as $t \to \infty$, $\frac{d\hat{A}_H(t)}{dt}$ converges to the mean arrival rate. $\hat{A}_H(t)$ increases with a decreasing rate after t^*. This means that the probability that the average arrival rate exceeds the link capacity decreases for $t > t^*$. Consequently, there is no need to worry about anything other than the time scale for which the source rate still exceeds the link capacity. In other words, after a period of time, the probability that the average arrival rate exceeds the link capacity is negligible, hence the arrival model needs no longer to reproduce the source behavior for these time scales.

5.4 Simulation Analysis of Dynamic Time Scale

To test the effects of the dynamic time scale approach, a simulation scenario similar to the one in Figure 3.1 is used. However, this time, besides effective capacity, t_{slot} is also recalculated after every N measurements. In other words, t^*, *dynamic time scale* is estimated using the recent N measurements and $t_{slot} = t^*$ is taken as the measurement slot duration for the next N measurements.

t^* is estimated directly from (5.1). However, instead of the $(C-m)$ term in (5.1), we used $L * m$, where L is taken as a constant $L = (AllocationRatio) - 1$. The reason is

that we allocate capacity dynamically and do not have a constant C. Actually, we tried using the moving average capacity allocation instead of C, but this caused a multiplicative effect, such that, when capacity allocation increases, $C - m$ term in (5.1) decreases. But decreasing t_{slot} results in increased capacity allocation measurement in the next window, and this loop ends up having $t^* \approx 0$.

Table 5.1 shows the improvement of using a dynamic $t_{slot} = t^*$ against static t_{slot} choices (GA is used as the capacity allocation algorithm, and the P_{loss} target is set to 10^{-3} as in the previous simulations). As the mean rate increases, the performance of the static t_{slot} cases changes (the ratio decreases and P_{loss} increases), whereas the performance of the dynamic $t_{slot} = t^*$ case remains the same. This shows that on-line measurement-based algorithms with constant measurement intervals are heavily dependent on the incoming traffic's mean rate, whereas the ones with dynamic measurement intervals are more robust. Also, note that the particular performance figures for dynamic t_{slot} case in Table 5.1 are dependent on the value of L (we used $L = 1.5$). However, we confirmed that the choice of L does not affect the robustness of the algorithm, which makes the algorithm desirable by making the performance independent from the input traffic.

Table 5.1: Performance Metrics vs. Time Scale

Mean (Kbytes/s)		t_{slot} 0.08s	t_{slot} 0.4s	t_{slot} 2s	t_{slot} t^*
4	Ratio	4.751	2.657	1.745	1.915
	P_{loss}	0	0	0.000003	0.000391
20	Ratio	2.952	1.739	1.353	1.914
	P_{loss}	0	0.000009	0.000382	0.000696
100	Ratio	1.742	1.334	1.183	1.913
	P_{loss}	0.000010	0.000332	0.002649	0.000910

To illustrate the benefits visually, we generated a traffic trace of 1000 s, where the mean rate of arrival traffic between 200 and 800 s is 5 times the mean rate at the remaining intervals. Figure 5.1 shows the dynamic capacity allocations. Note that the allocation ratios in the static t_{slot} cases change in the region of traffic with high mean rate. But the allocation ratio remains roughly the same in the dynamic time scale case. This is achieved by adjusting t_{slot} as shown in Figure 5.3.

The number of packets dropped increases when the mean is increased gradually in

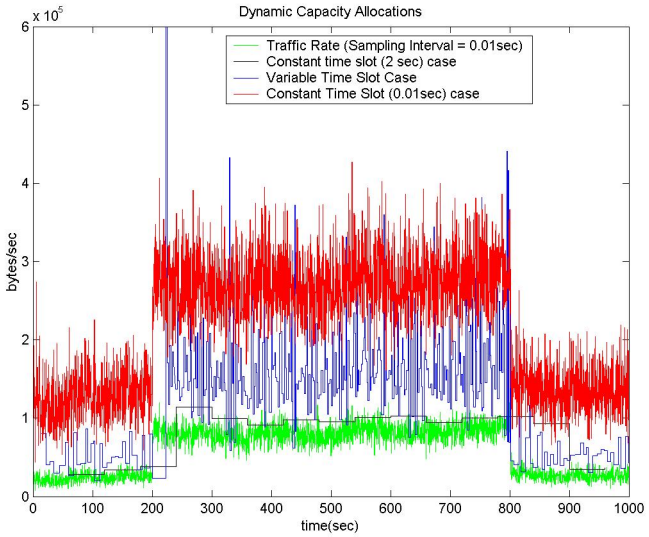

Figure 5.1: Capacity allocations vs. traffic mean rate.

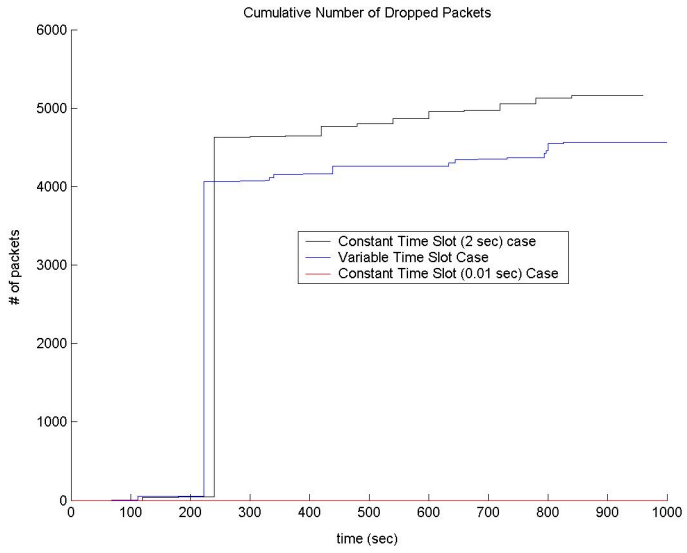

Figure 5.2: Cumulative number of dropped packets.

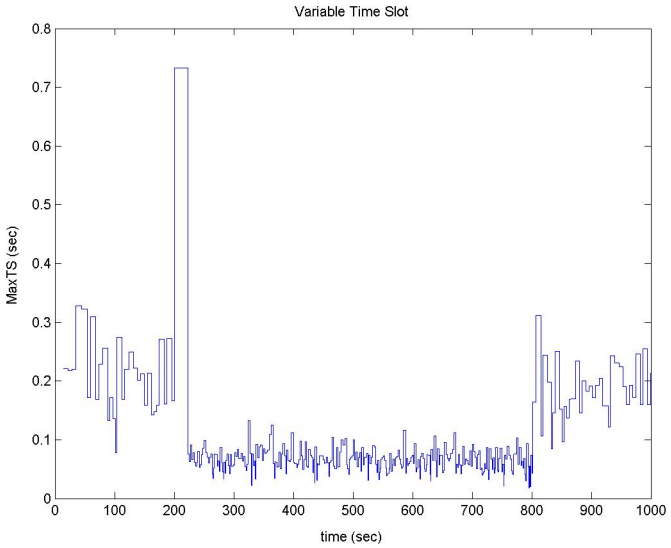

Figure 5.3: The plot of dynamic time scale parameter t^*.

Figure 5.2 (for $t_{slot} = 0.01$ s, no loss occurred, due to over-allocation). The t^* case performs again in between the static t_{slot} cases. But note that when $t_{slot} = t^*$, the algorithm can self-adjust and perform similarly against traffic mean changes, whereas the performance of an algorithm with static t_{slot} is dependent on the traffic. To illustrate, a method with static $t_{slot} = 0.01$ s case will over-allocate significantly when the mean rate decreases much below of 4 Kbytes/s, and a method with static $t_{slot} = 2$ s case will suffer significant degradation of the QoS target when the mean rate increases much above 100 Kbytes/s.

We left the the real implementation of this dynamic time scale approach as future work. As a final remark, note that in a real-world scenario, the measurement time scale has a lower threshold bound which comes from the computation complexity of the used algorithm. Other than that, we believe that this approach can easily be put into practical world, similar to the emulation experiment given in chapter 4.

5.5 Summary

In this chapter, we reason the need for dynamically changing the measurement time scale. We start with investigating how the measurement-based algorithm depends on the measurement time scale. Then, we argue that the measurement time scale is related to the arrival traffic. To illustrate, we show that when the traffic mean rate changes, one needs to change the measurement time scale accordingly, in order to keep the performance constant. At this point, we present our dynamic time scale. Finally, we provide simulation results and the benefits of this dynamic time scale approach.

Chapter 6

Conclusion

This thesis presented and compared measurement-based on-line capacity allocation algorithms and proposed a way to improve their robustness.

We distinguished such algorithms from MBAC and traffic predictors due to their smaller time scales and QoS-oriented use. We observed that their performance is directly dependent on the involved measurement time scales. Mainly, we saw that when the time slot length is increased while the window size is kept constant, due to increasing aggregation of packets in the slot interval, the variations between the measurements in the measurement window decrease and the allocated capacity approaches the mean traffic rate. This causes the loss probability to increase.

Since the measurement time scale is directly related to the measured traffic, the result of a measurement-based algorithm using constant time scale is open to the performance degradations due to the changes in traffic trends. However, our aim was to obtain an algorithm which does *not* require any *a priori* traffic knowledge, and which is based fully on the measurements. Therefore we incorporated the Maximum Time-Scale (MaxTS) parameter and tested successfully adapting the measurement time scales based on measurements themselves.

We performed a practical multiple stream effective bandwidth estimation and online link sizing experiment. The algorithms performed as suggested by the simulation results, giving us confidence in the feasibility of the algorithms. The real implementation can be improved by omitting the NeTraMet software which causes overhead in taking measure-

ments. Instead, a core application using packet matching library in an operating system would be much faster. Moreover, the application works for a constant measurement time scale and window size values. The implementation of the dynamic time scale presented in chapter 5 would make the real implementation robust against different traffic sources than the one used in chapter 4.

The outcomes of this study can be used for choosing algorithms to be implemented in real switches, taking into account trade-offs of complexity, accuracy and robustness.

To sum up, in this thesis, we

- identified on-line measurement-based capacity allocation algorithms,
- compared their performances analytically,
- simulated promising ones,
- observed significant affects of the choice of measurement time scale,
- proposed to vary measurement time scale adaptively,
- through an example, showed the performance robustness of measurement-based algorithms, in which measurement time scale is adaptive (*measurement-based*).

This thesis suggests the following future work:

- Addition of on-line Hurst parameter estimation as described in section 2.2.5.
- Analysis of the effects of different long-range dependent traffic on the algorithm's performance (we only used traffic with $H = 0.7$).
- Addition of the dynamic time scale approach (chapter 5) into the practical implementation (chapter 4).
- Decreasing the overhead in the time complexity of the real implementation by bypassing the NeTraMet software and using specialized packet sniffers based on core packet matching library in an operating system.

Bibliography

[1] L. Breslau, S. Jamin, and S. Shenker. Comments on the performance of measurement-based admission control algorithms. In *Proc. of IEEE Infocom '00*, volume 3, pages 1233–1242, Tel-Aviv, Israel, March 2000.

[2] C. Courcoubetis, V. A. Siris, and G. Stamoulis. Application of the many sources asymptotic and effective bandwidths to traffic engineering. *Telecommunication Systems*, 12(2-3):167–191, 1999.

[3] C. Courcoubetis and R. Weber. Buffer overflow asymptotics for a buffer handling many traffic sources. *Journal of Applied Probability*, 33:886–903, 1996.

[4] P. Droz. Wavelet-based resource allocation in atm networks. In *Proc. of IEEE Globecom '97*, volume 2, pages 833–837, Phoenix, Arizona, November 1997.

[5] N.L.S. Fonseca, G.S. Mayor, and C.A.V Neto. On the equivalent bandwidth of self-similar sources. *ACM TOMACS*, 10(2):104–124, April 2000.

[6] R.J. Gibbens and P.J. Hunt. Effective bandwidths for the multi-type uas channel. *Queueing Systems*, 9:17–28, 1991.

[7] M. Grossglauser, S. Keshav, and D. N. C. Tse. RCBR: A simple and efficient service for multiple time-scale traffic. *IEEE/ACM Transactions on Networking*, 5(6):741–755, 1997.

[8] M. Grossglauser and D. N. C. Tse. A framework for robust measurement-based admission control. *IEEE/ACM Transactions on Networking*, 7(3):293–309, 1999.

[9] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal of Selected Areas in Communications*, 7(7):968–981, 1991.

[10] Q. Hao, S. Tartarelli, and M. Devetsikiotis. Self-sizing and optimization of high-speed multiservice networks. In *Proc. of IEEE Globecom '00*, volume 3, pages 1818–1823, San Francisco, November 2000.

[11] http://lartc.org/HOWTO. Linux advanced routing and traffic control howto.

[12] C. Huang, I. Lambadaris, M. Devetsikiotis, P.W.Gylnn, and A.R. Kaye. DTMW; a new congestion control scheme for long-range dependent traffic. In *Proc. of 15th ITC*, Washington, D.C., June 1997.

[13] F.P. Kelly. *Stochastic Networks: Theory and Applications*, pages 141–168. Oxford University Press, 1996.

[14] G. Kesidis, J. Walrand, and C. S. Chang. Effective bandwidths for multiclass markov fluids and other atm sources. *IEEE/ACM Transactions on Networking*, 1(4):424–428, August 1993.

[15] S. Li, S. Park, and D. Arifler. SMAQ: A measurement-based tool for traffic modeling and queueing analysis, part I - design methodologies and software architecture. *IEEE Communications*, 36:56–65, August 1998.

[16] S. Molnar and T. D. Dang. Pitfalls in long range dependence testing and estimation. In *Proc. of IEEE Globecom '00*, San Francisco, November 2000.

[17] D. Morato, J. Aracil, L. A. Diez, M. Izal, and E. Magana. On linear prediction of internet traffic for packet and burst switching networks. In *Proc. of ICCCN '01*, pages 138–143, Scottsdale, Arizona, October 2001.

[18] I. Norros. On the use of fractional brownian motion in the theory of connectionless networks. *IEEE Journal of Selected Areas in Communications*, 13(6):953–962, 1995.

[19] S.A.M. Ostring, H.R. Sirisena, and I. Hudson. Rate control of elastic connections competing with long-range dependent network traffic. *IEEE Transactions on Communications*, 49(6):1092–1101, June 2001.

[20] J. Qiu and E. W. Knightly. Measurement-based admission control with aggregate traffic envelopes. *IEEE/ACM Transactions on Networking*, 9(2):199–210, 2001.

[21] M. Roughan, D. Veitch, and P. Abry. Real-time estimation of the parameters of long-range dependence. *IEEE/ACM Transactions on Networking*, 8(4):467–478, 2000.

[22] B. K. Ryu and S. B. Lowen. Point process approaches to the modeling and analysis of self-similar traffic, I. model construction. In *Proc. of IEEE Infocom '96*, volume 3, pages 1468–1475, San Francisco, March 1996.

[23] W. Shen. On-line measurement of effective bandwidth and a hierarchical self-sizing framework. Master's thesis, Institute for Electrical and Computer Engineering, Carleton University, April 2002.

[24] S. Tartarelli, M. Falkner, M. Devetsikiotis, I. Lambadaris, and S. Giordano. Empirical effective bandwidths. In *Proc. of IEEE Globecom '00*, San Francisco, November 2000.

[25] D. Veitch and P. Abry. A wavelet based joint estimator of the parameters of long-range dependence. *IEEE Transactions on Information Theory*, 45(3):878–897, 1999.

[26] J. Yan. Adaptive configuration of elastic high speed multiclass networks. *IEEE Communications*, 36:116–120, May 1998.

[27] J. Yang and M. Devetsikiotis. On-line estimation, network design and performance analysis with effective bandwidths. In *Proc. of 17th ITC*, Salvador Da Bahia, Brazil, December 2001.