Distributed and Scalable Voice Admission Control Algorithm for Over-subscripted Networks

Parag Kulkarni[1,2], Petre Dini[2], Sally McClean[1], Gerard Parr[1], Michaela Black[1]

[1]School of Computing and Information Engineering

University of Ulster, Coleraine, Northern Ireland BT52 1SA.

Email: {pg.kulkarni, si.mcclean, gp.parr, mm.black}@ulster.ac.uk

[2]Cisco Systems Inc., San Jose, California 95134, USA

Email: pdini@cisco.com

Abstract: The idea of carrying voice traffic on IP networks has been found to be very lucrative. However to achieve a quality like the one offered by the existing telephone networks, the IP network should be in a position to honour the stringent Quality of Service (QoS) requirements of voice traffic. If traffic more than what the network can handle is admitted into the network, QoS may be violated resulting in performance degradation. One way of providing sustained and consistent QoS is by regulating the number of voice calls that are admitted into the network such that the load on the network is less than or equal to the capacity of the network. This mechanism is called call admission control. This paper outlines the design and simulation of a scalable and distributed call admission control algorithm that operates at the network edge and factors in the local passive measurements in the admission decision. Performance evaluation of the proposed algorithm through ns2 simulations reveals that it regulates the number of calls in the system around the maximum admissible limit thereby delivering on the QoS guarantees demanded by voice applications. Moreover, it is simple and lightweight from an implementation perspective.

Keywords: Voice over IP, Admission control, over-subscripted network services.

Parag Kulkarni graduated with a B.Tech degree in Computer Science and Engineering from the Regional Engineering College (REC) Calicut, India in 2001. He then joined eInfinitus Technologies at the IIT Bombay Business Incubator as a Member of Technical Staff where he was involved in the development of algorithms to support dynamic QoS in MPLS networks. Since October 2003, he is a Doctoral student in Computer Science at the University of Ulster. During the summer of 2006, he was a visiting research intern at Cisco Systems in San Jose California. His research interests lie in the areas of proactive network management, QoS management in IP and MPLS networks, predictive modeling for traffic management and adaptive learning. He is a student member of the IEEE.

Prof. Dr. Petre Dini is now with Cisco, USA, as a senior technical leader and principal architect, being responsible for policy-based strategic architectures and protocols for network management, QoS, SLA, and performance, programmable networks and services, provisioning under QoS constraints, wireless networks and protocols, and consistent service manageability. His applied industrial research interests include instrumentation software agents, performance, scalability, autonomic computing, wireless and mobile networks, constraints in wireless networks, adaptive networks, sensor networks, and policy-related issues in adaptable networks. He led various Canadian projects related to object-oriented management applications for distributed systems, and to broadband services in multimedia applications, until early 1996. In 1996 he joined Computer Science Research Institute of Montreal and coordinated many projects on distributed software and management architectures. In this period he was an Adjunct Professor with McGill University, Montreal, Canada, and a Canadian representative in the European projects. Since 1998 he was with AT&T Labs, as a senior research manager, focusing on distributed QoS, SLA, and performance in content delivery services. He was the Co-Chair of Policy-Based Management Work Group in Telemanagement Forum, is a Rapporteur for ITU-T/SG4, and actively involved in IEEE industrial initiatives. He has been an invited speaker to many international conferences, a tutorial lecturer, and chaired several international conferences. He published more than 100 papers in prestigious conferences and journals, and has more than 20 granted or pending patents He is an adjunct professor at Concordia University, Canada.

Sally McClean is a Professor of Mathematics at the University of Ulster. Her research interests are in statistical modelling, and applications to computer science. Her first degree was in Mathematics at Oxford, a M. Sc. in Cardiff and Ph.D. at the University of Ulster. She has published and edited five books and over 200 papers. She is a member of the IEEE.

Gerard Parr holds the Full Chair in Telecommunications Engineering and is the coordinator of the Internet Technologies Research Group in the Faculty of Engineering at the University of Ulster, Coleraine campus, Northern Ireland. He holds a PhD in Self Stabilizing Protocols since 1990. He is a member of the UK EPSRC College (Communications Panel) and sits on the Editorial Board of Elsevier Journal of Computer Networks and the International Journal of Aerospace Computing. He has also served as a TPC member on several IEEE conferences including Policy 04, MILCOM 04, MMNS 05, IPOM 05, CCNC 05 and WCNC 05.

Michaela Black holds a PhD in machine learning from the University of Ulster. Her PhD was partly funded by BT Research Labs, Ipswich and the CASE award. Her PhD addressed the problem of detecting and adapting to concept drift in rapidly changing data, and there are patents pending on some of the algorithms that she developed as a part of this work. Her research interests are in the areas of artificial intelligence, data mining and machine learning. She has been applying some of the insights that have emerged from her research to such interdisciplinary fields as telecommunications, bioinformatics and digital games.

1. Introduction

Voice over IP (VoIP) is gaining popularity due to its cost benefits. This is evident from the push exhibited by enterprises and service providers to migrate their infrastructures towards a converged IP based network. However, to provide any meaningful VoIP services, the network must honour the stringent QoS requirements of voice traffic - a one way delay of not more than 200ms, loss rate of 1% or less and an average jitter less than equal to 50ms (see [1] for details). If more connections[1], than what the network can cope with, are admitted into the network, the QoS of already admitted connections may be violated (increased delay/loss/jitter) and may lead to performance degradation. Typically, the number of subscribers out-numbers the network capacity (or in other words, the scarce network resources are over-subscribed). Thus, what is desired in such circumstances, is an admission control mechanism that can regulate the number of connections in the network so as to provide acceptable QoS to each voice call admitted into the network.

Several approaches to admission control have been investigated to date and have resulted in some valuable contributions such as [2] [3] [4] [5] [6] [7] [8] [9]. Some of these efforts have focused on devising an approach that employs signalling and resource reservation on a hop by hop basis (e.g. [2] [7] [10]). A comprehensive review of algorithms belonging to this category can be found in [6] [11]. These algorithms typically comprise of two phases at each hop – the measurement phase (where the current load at that hop is estimated) and the admission phase (where the estimated load drives the admission decision at that hop). These approaches provide a fine grained QoS by maintaining per-flow state on the end to end path. However, maintaining per-flow state is undesirable as it leads to scalability problems. Hence, despite being in a position to honour QoS guarantees of individual flows, these approaches are not feasible due to their inability to scale.

Bandwidth broker based approaches have also been proposed in [12] [13] [14] where a centralized agent in the domain takes admission decisions. To achieve this, it maintains a topology database and keeps track of the available bandwidth on each link in the network. Whenever a connection request arrives, it admits this request if the requirement of this new request is less than the available capacity and rejects the request otherwise. This approach does not scale due to its centralized architecture and is also susceptible to being the central point of failure. Moreover, the cost of updating the topology database may result in a huge communication overhead if the available bandwidth at the links keeps changing frequently.

There also exist other approaches that have adopted a different paradigm wherein instead of reserving the resources, the end point or the edge router probes the network and depending on the outcome of the probing process (% of probe packets lost, delay experienced by the probe packets etc.) takes the admission decision. [3] [4] [5] are some of the main approaches that subscribe to this school of thought. A thorough review of approaches belonging to this category can be found in [5]. The main merit of approaches in this category is that no per flow state needs to be maintained on the end to end path and hence these are scalable. They however, suffer from a few problems as identified by the study in [5]. Firstly, if too many flows are probing the network at the same time, all of them could be rejected despite the fact that a few could have been admitted. This is because all the probes might experience very high loss/delay during the probing process. Network based admission control approach such as that employing RSVP signalling does not suffer from this problem because requests are serialized. In such conditions the RSVP approach would ensure that a few flows are admitted. Secondly, the probing approach may not suit real time applications as there is a setup delay of a few seconds that is incurred during the probing process. Finally, users may not have the incentive to perform admission control and hence may send traffic without probing thereby causing congestion.

In [8], the authors propose a seminal approach to distributed admission control based on the core-stateless principle where the ingress router takes the admission decision. The main idea is to treat all the edge routers of a network domain as a logical token ring. A token that contains information about the available capacity of each link in the network is circulated between all the edge routers one by one. Connection requests may arrive at any or all of edge routers at any time. However, only the edge router which has the token, can decide the fate of requests (if any) that have arrived at it and are pending a decision. After taking the admission decision, this edge router updates the available capacity in the token by subtracting the resources demanded by this request from the available capacity on the edge to edge path through which this request will traverse. After processing all admission requests that are pending at itself, it then forwards the token to the next edge router so that this router can process admission requests that have arrived (and are pending) at it. Despite being scalable, the approach has a significant shortcoming. The

[1] In this paper, the terms 'call' and 'connection' are used interchangeably

ingress router has to wait until it gets the token in order to take admission decisions. This introduces a setup delay which might be unacceptable to real time traffic such as VoIP. Moreover, edge routers require co-ordination amongst themselves (only one edge router can take admission decision at a time) as opposed to our proposed approach where each edge router takes admission decisions independently in a truly distributed manner.

Our approach is inspired by the one in [9] where admission requests are processed at the egress router. Our approach is, however significantly different from this proposal in that it employs a simple average of the delay over the past 'n' observations on each packet arrival to take an admission decision as opposed to their approach of computing complex traffic envelopes. Moreover, our approach is extremely lightweight and is primarily geared towards voice traffic. It is also very different from the other approaches mentioned in the previous paragraphs. It uses passive measurement of delay at the exit point of the network (through which the connection request passes) to take admission decisions. Unlike its hop-by-hop measurement based counterparts, it processes the signalling request only at the exit gateway of the network. Unlike its bandwidth broker based counterparts, it is scalable and distributed and finally, unlike its probe based counterparts, it does not incur a setup delay. It is simple and renders itself to a lightweight implementation.

2. Preliminary Approach and Experiments

To start with, we decided to explore the feasibility of a prediction based approach similar to the one outlined in [15][2]. The main motivation behind exploring such an approach was driven by the desire to be able to foresee the delay in the near future and pro-actively reject incoming admission requests before QoS is violated. The underlying principle was to sample the instantaneous delay every 'SI' seconds on each ingress-egress path. This interval is termed the 'Sampling Interval (SI)'. There is also a related interval namely 'Prediction Interval (PI)' which is comprised of a number of sampling intervals i.e. PI/SI sampled observations of the instantaneous delay on each ingress-egress path. At the start of each PI, the average (arithmetic mean) of the sampled delay observations over the previous prediction interval is computed. This average value is then used to predict the average delay over the next prediction interval. The Recursive Least Squares (RLS) algorithm is used to compute this prediction as an adaptive learning function of the previous average delays. The deviation of predicted delay from the RejectionThreshold (explained in detail in section III B) drives the admission decision. If predicted average delay is greater than RejectionThreshold, admit bit is set to 0 indicating that all flows requesting admission over the next interval should be denied admission. On the other hand, if predicted average delay is less than Rejection Threshold, admit bit is set to 1 indicating that all flows requesting admission over the next interval should be accepted. Note that the admit bit is set only once at the beginning of each PI and stays the same throughout that interval. On every packet arrival, if packet is of type admission REQUEST, then the request is admitted if admit bit is 1 otherwise the request is rejected. Algorithm 1 summarizes the prediction based admission control algorithm.

Algorithm 1 Predictive Admission Control Algorithm

```
Input - average delay over the past interval
for every PI seconds do
    Compute average delay over the past interval
    Predict average delay over the next interval - execute algorithm 2
    if predicted delay > RejectionThreshold then
        set admit bit to 0
    else
        set admit bit to 1
    end if
end for
On Every Packet Arrival
if packet is of type REQUEST then
    if admit bit == 0 then
        Reject the request
    else
        Admit the request
    end if
end if
```

[2] [15] has elaborated upon a prediction based queue management approach

A. Predictor Algorithm

1) *Choice of a predictor*: Adaptive filters are a well known class of online estimation techniques; one of their applications being prediction. Simplicity, accuracy, adaptability and computational overhead were the key objectives underlying the choice of a predictor. As will be evident from the ensuing discussion, the model employed in this paper, RLS algorithm, is simple, computationally efficient and delivers a very good prediction accuracy. Moreover, as network traffic is highly dynamic, it is desirable to have an algorithm that can adapt quickly. Since the adaptation time of RLS is small (equal to 2N where N is the number of past samples used in prediction), it becomes the obvious choice.

TABLE I: NOTATION USED IN THE RLS ALGORITHM

n	Current step
N	Number of past observations used to predict the future
kStep	Number of steps ahead in the future for which prediction will be made
FF_λ	Forgetting factor
I	Identity Matrix (N x N matrix)
K(*n*)	Updating Step Size. Also called Gain Vector (N x 1 matrix)
P(*n*)	Inverse of the Input Autocorrelation matrix (N x N matrix)
avgD(*n*)	Input vector comprising of average delay observations over the past N prediction intervals (N x 1 matrix)
w(*n*)	Weight vector comprising of weights associated with past observations of $avg_d(n)$ (N x 1 matrix)
predD(*n*)	Predicted value of average delay over the next interval (Scalar term).
e(*n*)	Error vector (N x 1 matrix), where e(n) = avgD(n+1) - predD(n)

TABLE II: INITIAL VALUES FOR THE PARAMETERS OF THE RLS ALGORITHM

kStep	1
FF_λ	0.95
P(*n*)	δ * I (where δ = small +ve constant, 100 in this case)
w(*n*)	weight vector is initialised to 0
e(*n*)	error vector is initialised to 0

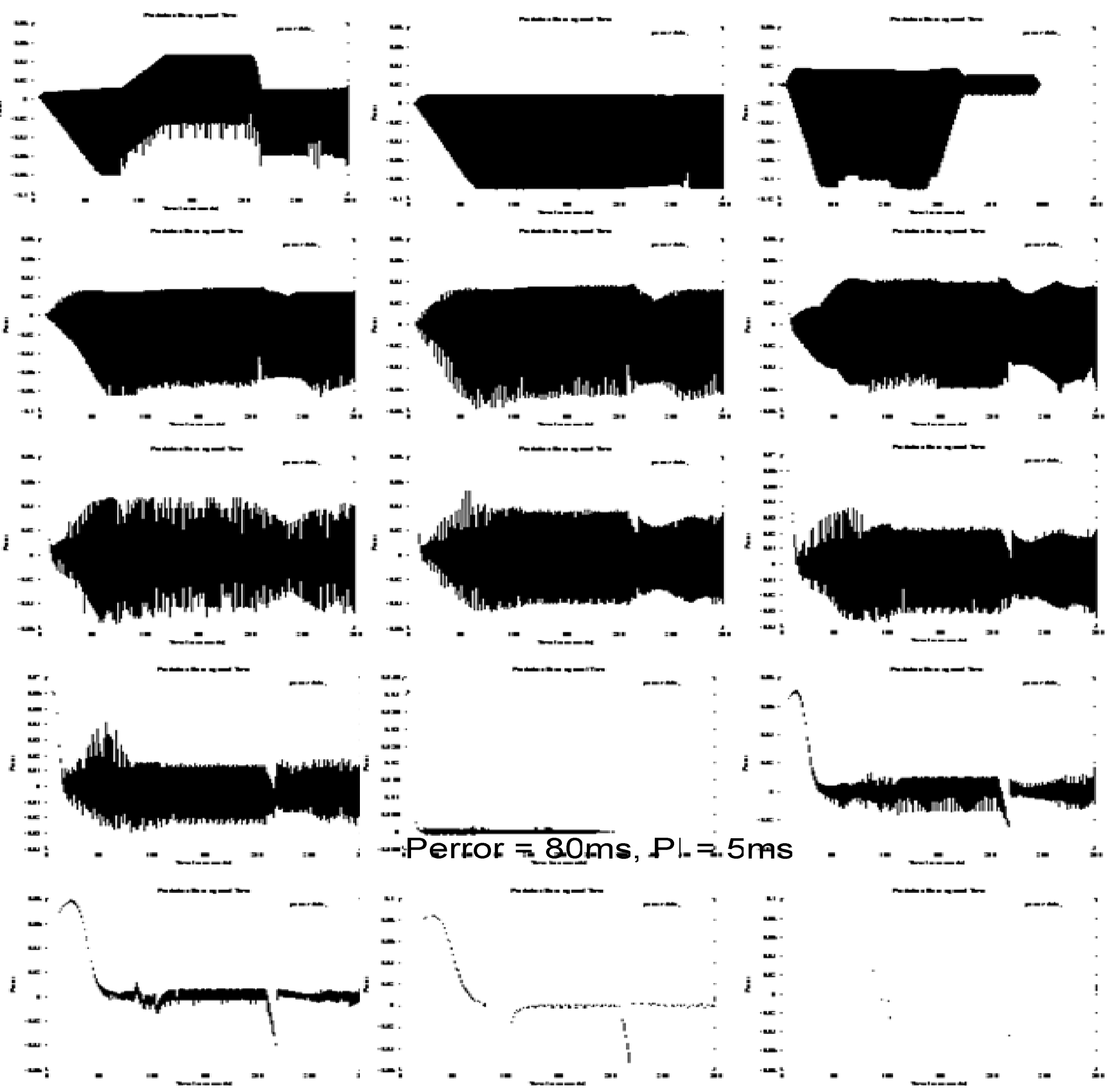
2) RLS Algorithm: The model which describes the system under consideration is the RLS algorithm and has several associated parameters which are shown in Table I. Some of these parameters are static (*N*, *kStep*, *FF*,) and the others are dynamic (*K*(*n*), *P*(*n*) and *w*(*n*)). The dynamic parameters are estimated online during system operation and keep adapting to changes in the underlying system (adaptive learning). This process is called recursive parameter estimation and forms the key principle behind the RLS algorithm whose main objective is to minimize the sum of squared error.

Algorithm 2 Steps involved during each iteration of the RLS Algorithm

```
Step 1. Compute the gain vector.
        temp(n) = P(n-1)*avgD(n)
        $K(n) = \frac{temp(n)}{FF_\lambda + avgD^T(n) * temp(n)}$
Step 2. Compute the predicted average delay
        $predD(n) = w^T (n-1)*avgD(n)$
Step 3. Compute the prediction error
        e(n) = avgD(n) - predD(n-kStep)
```

```
Step 4. Adapt the weight vector
        $w(n) = w(n-1) + K(n)*e(n)$
Step 5. Update the P matrix
        $P(n) = \frac{P(n-1) - K(n)*temp^T(n)}{FF_\lambda}$
```

Table I explains the notation used in the RLS Algorithm and Table II shows the initial values used for the various parameters of the RLS algorithm. The choice of initial values for the RLS algorithm has little or no effect on the accuracy of predictions. Algorithm 2 summarizes the steps involved during each iteration of the RLS algorithm; note that Step 2 is the prediction phase whereas Step 4 is the online learning phase, where the prediction error (difference between the actual and predicted average delay) computed in Step 3 is used to dynamically update the weight vector. As shown in Step 2 of algorithm 2, the predicted average delay is calculated as a weighted sum of the observed average delay values over the past *N* intervals. In line with the approach in [15], we chose N=1. To choose a suitable value of PI which delivered accurate predictions, we ran several experiments for values of PI ranging from 5ms to 100ms. Fig.1 shows the prediction error for different values of PI. It is evident from these figures, that the prediction delivered by the RLS algorithm is better for higher values of PI. The best prediction accuracy (prediction



Perror = 80ms

Figure 1. Prediction error v/s time for different values of PI

error less than 5ms on either side of 0) is obtained for PI=100ms. The plot corresponding to PI=100ms, is perfectly in accordance with the behaviour of the RLS algorithm in that the prediction error sharply

Perror = 80ms, PI = 30ms

Perror = 80ms

increases at the beginning and then converges to 0 with time. A high initial error is attributed to the fact that the algorithm does not have any history information. However, as time goes by, more of the historical information is available to the algorithm to compute a prediction because of which it makes less mistakes which result in a low prediction error. Thus we can conclude that in order to capture the fluctuations in the underlying system (or in other words to exploit predictability in the underlying system), prediction should be computed every 100ms.

B. Problems with this Approach

The key shortcomings of the prediction based approach were that the best predictions were delivered as we increased the size of the interval. This implied that the admit/no-admit decision affected all flows in the following interval i.e. all the flows requesting admission over the next interval would all be either accepted or rejected. All flows over the next interval being admitted may lead to more flows being admitted than the capacity resulting in QoS violations. On the other hand, no flows in the next interval being admitted may lead to number of flows admitted into the system being less than the capacity resulting in wastage of network resources. Hence an interval based approach is not appropriate in this context. It is important to track changes in delay occurring at the packet level so that admissions can be rejected as soon as packets start experiencing more delay rather than waiting for a time interval to elapse. To achieve this, we developed a simple algorithm that works on the packet level time scale. The finer aspects of this algorithm and its performance evaluation are the subject of the remaining sections in this paper.

3. Algorithm

Some assumptions have been made to simplify the discussion. It should however be noted that these assumptions do not have any bearing on the performance of the algorithm. These assumptions are as follows: Each flow (voice call) will explicitly initiate call setup using a signalling protocol (e.g. Resource Reservation Protocol - RSVP [16] employed in this work). Similar to the approach in [9], RSVP signalling messages are processed only at the Egress router (hence the admission decision is made at the Egress router) and the core routers provide the same forwarding treatment to these packets as any other packet within the same traffic class. Each voice call is carried in the same traffic class with an end to end target delay of this class being 150ms[3]. Sum of the fixed delays i.e. coding delay at the source and, propagation and serialization delay at each network link on an end to end (e2e) path is less than the target delay of 150ms. If this is not the case, then no algorithm can meet the e2e delay guarantee of 150ms. Given this, the amount of traffic to be admitted such that the delay guarantees are met is then the job of the admission controller. Source inserts a time-stamp in each packet before sending it which is then used to compute the delay along that ingress-egress path. Note that if no changes are desired at the end host then this task of inserting a time-stamp in the packet can be performed at the ingress router. Additionally, each ingress inserts its identifier in every packet that is forwarded into the network. It is assumed that clocks are synchronized. The problem of clocks being out of synchrony can be addressed using existing solutions as outlined in [9].

A. Key Idea Central to the Algorithm

During the journey of a packet belonging to a voice call from the source to its destination it encounters several types of delays - e.g. coding delay at the source (codec), serialization delay and propagation delay at the links on its path, all of which are fixed delays. Additionally, it may or may not encounter queuing delay (which may be variable) depending on the state of congestion in the network. In the scenario of no congestion in the network, the delay suffered by any packet by the time it reaches the exit point of the network is the sum of all the aforementioned fixed delays on its path from the source. Whenever the packet delay starts rising above this, it is an indication that the net rate of traffic entering the network destined for this exit point is greater than the rate of traffic leaving this exit point. In other words, traffic is being buffered somewhere in the network along the path leading to this exit point. This is the point at which the admission control mechanism should start refusing further admissions into the network that pass through this exit point. This is key idea behind the algorithm which we will elaborate upon in the next subsection.

[3] This value of target delay has been chosen in line with requirements of voice traffic as outlined in [1].

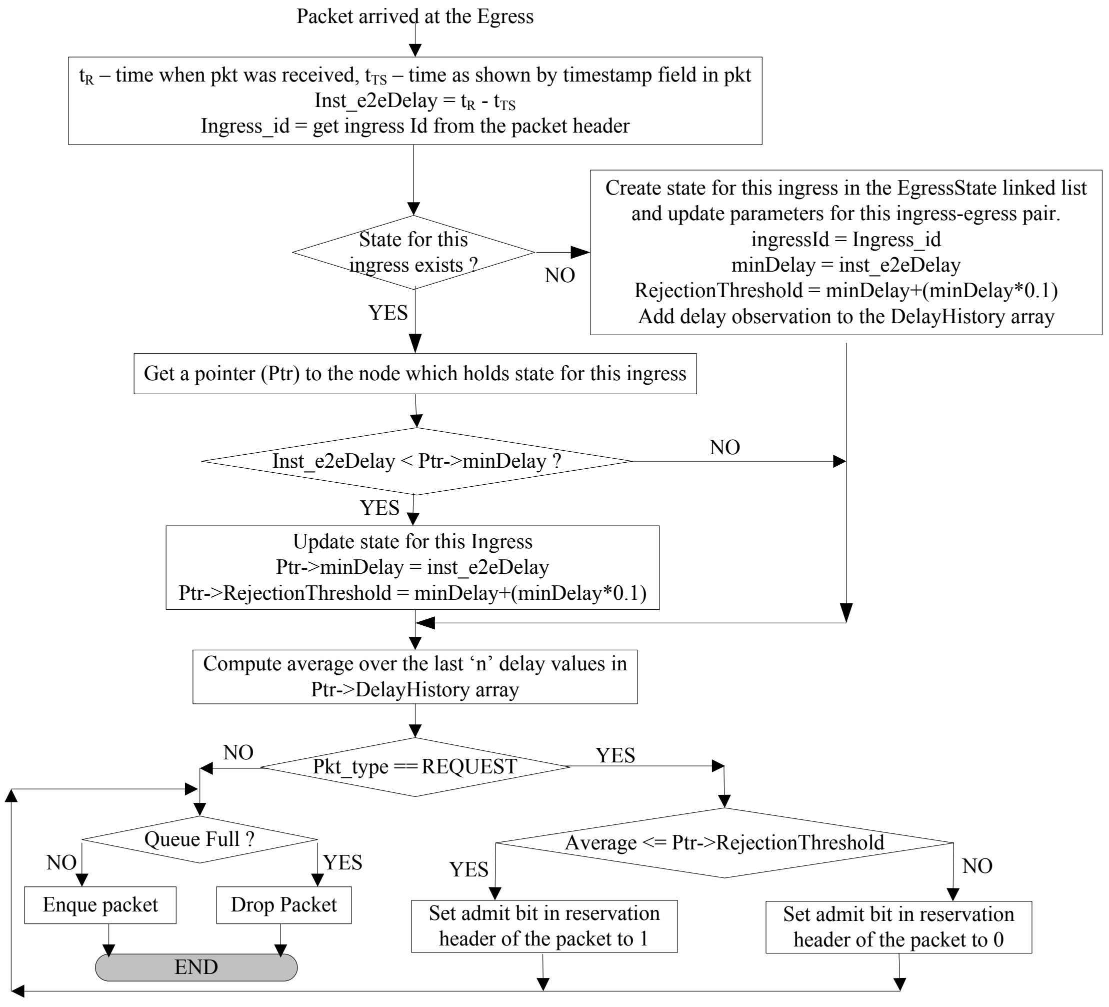

Packet arrived at the Egress
t_R – time when pkt was received, t_{TS} – time as shown by timestamp field in pkt
Inst_e2eDelay = t_R - t_{TS}
Ingress_id = get ingress Id from the packet header
State for this ingress exists ?
NO
YES
Create state for this ingress in the EgressState linked list and update parameters for this ingress-egress pair.
ingressId = Ingress_id
minDelay = inst_e2eDelay
RejectionThreshold = minDelay+(minDelay*0.1)
Add delay observation to the DelayHistory array
Get a pointer (Ptr) to the node which holds state for this ingress
Inst_e2eDelay < Ptr->minDelay ?
NO
YES
Update state for this Ingress
Ptr->minDelay = inst_e2eDelay
Ptr->RejectionThreshold = minDelay+(minDelay*0.1)
Compute average over the last 'n' delay values in Ptr->DelayHistory array
Pkt_type == REQUEST
NO
YES
Queue Full ?
NO
YES
Enque packet
Drop Packet
END
Average <= Ptr->RejectionThreshold
YES
NO
Set admit bit in reservation header of the packet to 1
Set admit bit in reservation header of the packet to 0

Figure 2. Admission Control Logic

B. Admission Control Logic

Each source initiates call setup by sending a 'REQUEST' packet which contains information pertaining to the rate at which the source will be sending traffic (64Kbps CBR in this case assuming a G.711 voice codec). This REQUEST packet travels all the way up to the egress gateway connected to the destination. The egress gateway on receipt of this packet processes it in accordance with the admission logic which is outlined in Fig. 2. It should be noted that the call setup REQUEST packet is the only packet processed by the egress gateway and all other packets are simply forwarded as is to the next hop. Based on the admission decision, the egress gateway sets the admit bit in the reservation header of the REQUEST packet to either 1 (indicating accept) or 0 (indicating reject). On receipt of this REQUEST packet, the destination either sends an ACCEPT packet or a REJECT packet to the source depending on the value of the admit bit in the reservation header of this REQUEST packet.

1) Rejection Threshold: As mentioned earlier, when the packet delay starts rising above the minimum delay on the path to an exit point, this indicates a mismatch between the input traffic rate (at ingress) and output traffic rate (at egress). This is the point at which further admissions destined for this particular egress should be stopped from this ingress. This is achieved by employing a threshold called *Rejection Threshold*. Rejection threshold (one threshold for each ingress-egress pair), is the value of delay, which when exceeded at an egress node results in further connection requests (passing through this egress node) being rejected by the network. In order to prevent the admission control algorithm from being highly sensitive to even a minute change in delay, this value has been set to 10% more than the minimum end to end delay as shown in Fig 2.

2) Need for averaging and impact of 'n': At each egress, the last 'n' values of packet delay, as seen on the path between each ingress and itself is maintained. On receipt of a new connection request, in order to make the admission decision it is necessary to identify if a trend corresponding to increasing packet delays has been observed. This is achieved by computing the average (arithmetic mean) of the last ‘n’ packet delay values for this ingress-egress pair. If the average value is less than the Rejection threshold, then this connection request is accepted otherwise it is rejected. The averaging operation helps to prevent the algorithm from being sensitive to minor changes in delay experienced by consecutive packets by smoothing out any haphazard variations. The value of ‘n’ is an indicator of the amount of history used to identify a trend. A larger value of ‘n’ implies that more time is required to identify a trend. This can lead to more connections being admitted into the network (than what the network can handle) before a violation is detected. This can have a bad impact on the performance of the algorithm. On the other hand, a lower value of ‘n’ helps to quickly identify a trend If we assume that all of the last ‘n’ packets were signalling REQUEST packets, then the algorithm will admit more connections if the history is larger than when the history is smaller. Hence we recommend choosing a small value for ‘n’. The value of ‘n’ used in this work is 5.

C. Summary of the Salient Features of the proposed Algorithm

- **Simple and Lightweight:** The proposed algorithm is simple and renders itself to a lightweight implementation as its operation involves computation of a simple arithmetic mean on each packet arrival of the past ‘n’ packet delay values at the egress router.
- **Scalable:** The proposed approach is highly scalable as it does not employ any form of resource reservation (i.e. core stateless principle wherein no per flow state is maintained on the end to end path). RSVP signalling messages generated by the sources, are processed only at the edge of the network at the Egress routers and the core simply forwards these messages without any extra processing. The only state that each Egress router maintains is the past ‘n’ packet delay observations for packets arriving from each Ingress. If there are *N* edge routers, then the total amount of state maintained at each egress router is equal to *n*(N-1)*. Thus, the amount of state that needs to be maintained is low and is a linear function of the number of edge nodes.

4. Simulation Results

This section details the performance evaluation of the proposed admission control algorithm in a simulation setting. Towards this end, the proposed algorithm was implemented in the ns2 network simulator [17]. The proposed mechanism was evaluated in terms of the following performance metrics:

- Per path end to end delay as measured at the Egress router
- Number of voice calls (henceforth referred to as active flows) in the system at any point of time along each end to end path and
- Net loss ratio (ratio of total number of packets lost at the bottleneck queue(s) to the total number of packets that arrived at the bottleneck queue(s)).

A. Scenario 1: Single bottleneck link with homogeneous end to end delay

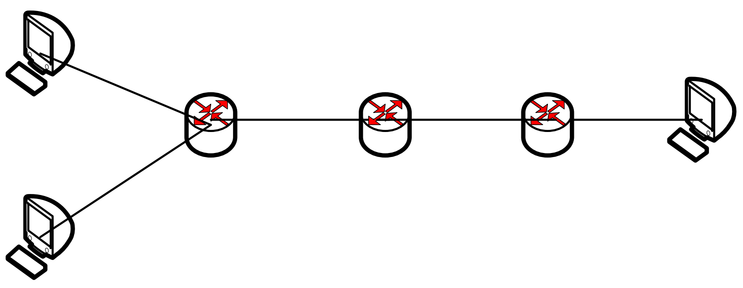


Figure 3. Single bottleneck, homogeneous e2e delay

Experiments under this scenario were carried out on the topology shown in Fig 3. As shown in this figure, traffic from both the sources encounters identical fixed delays to the egress of 40ms. Connection arrivals at the sources were exponentially distributed with mean arrival rate (λ) being 2 and 4 per second respectively during each of the two experiments. This results in subscription levels of approximately 75% and 150% of the network capacity respectively. In this scenario, the maximum number of calls that can simultaneously co-exist in the system without causing congestion are 25Mbps/64Kbps[4] (approx 390). Thus, under ideal circumstances, irrespective of the subscription level, the admission controller should regulate the number of simultaneously existing calls to less than or equal to 390.

B. Scenario 1: Results

Fig. 4 shows the end to end delay and number of active calls in the system against time for both the aforementioned experiments with λ = 2 and 4 respectively. As shown in figure, in the first case where λ = 2 (Fig.4(a)(c)), the number of calls in the network at any given point of time is less than the maximum number of calls that the network can handle. Thus, there is no congestion in the network because of which all calls requesting admission are successfully admitted into the network, the end to end delay remains equal to the sum of fixed delays on the path from the source to the destination and there is no packet loss. In the second case (λ = 4, Fig.4(b)(d)), the number of calls requesting admission in the network is greater than the capacity of the network as a result of which some calls have to be rejected.

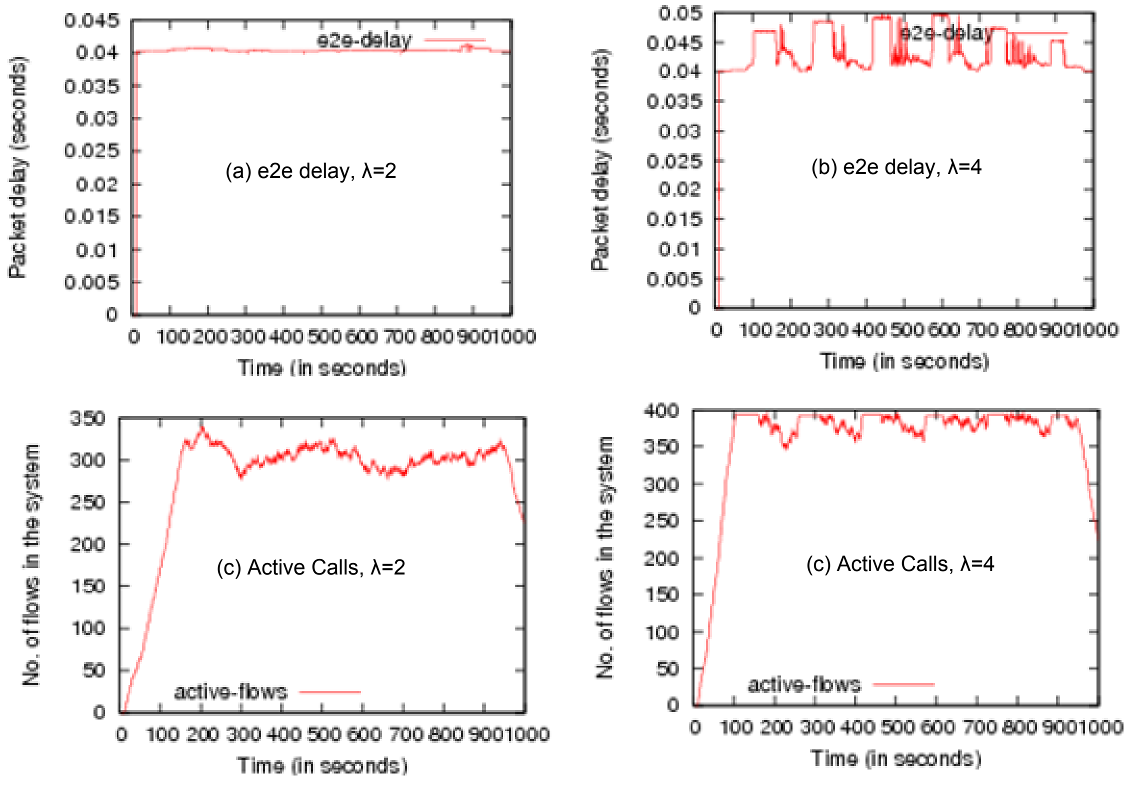


Figure 4. Scenario 1: Instantaneous e2e delay and number of active calls v/s time for each of the two experiments with λ = 2 and 4

The increase in delay (above the sum of fixed delay) is attributed to the admission of a few calls in excess of the maximum allowable limit which is 390 in this experiment. Once these extra calls are admitted, the delay remains high until the number of calls in the system decreases below 390. This is evident from the square shapes in the delay graph of Fig 4(b). Note that in the presence of these extra calls, packets may be lost because the buffer size at the bottleneck is kept quite small (60 packets)[5] in order to minimize the queuing delay. The loss ratio observed in this case was 0.23% which is very low and hence does not have any adverse impact on the performance of the algorithm. The reason why a few calls more than the capacity are admitted into the network is because we use the average over past 5 delay samples to ascertain if delay has crossed the Rejection Threshold. For the average to cross Rejection Threshold at

[4] Assuming the bandwidth requirement of each call to be 64Kbps

[5] A buffer size of 60 packets for a link bandwidth of 25Mbps results in a maximum queuing delay of approx 10ms.

least one or two delay observations out of five should be above the Rejection Threshold. This can happen only when there is at least one excess call in the system. It should be noted that more than 1 extra calls may be admitted into the network in the event of the average delay not growing beyond the Rejection Threshold and consecutive signalling request packets that are received at the same time being granted admission into the network. In this experiment, 4 excess calls were admitted because when the signaling packets for these calls were received, the average delay was below the Rejection Threshold.

C. Scenario 2: Multiple bottleneck links with heterogeneous end to end delay

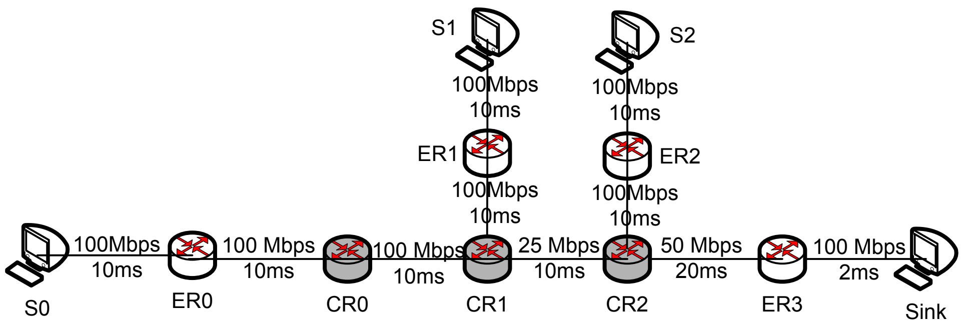


Figure 5. Multiple bottlenecks, heterogeneous e2e delay

The topology used to evaluate the performance of the algorithm under this scenario is shown in Fig 5. As shown in figure, traffic from sources S0, S1 and S2 encounters different fixed end to end delays of 60ms, 50ms and 40ms respectively. The bottleneck link CR1-CR2 whose capacity is 25Mbps (maximum capacity in terms of number of simultaneous calls being 390) is shared by traffic from the two edge routers ER0 and ER1 respectively. This link is the lowest capacity link on the end to end path for traffic from both of these edge routers. Hence the sum of active calls from each of the edge routers ER0 and ER1 should ideally not exceed 390. The CR2-CR3 link on the other hand is shared by traffic from all three edge routers ER0, ER1 and ER2. The capacity of this link is 50Mbps which approximately translates to a call capacity of 781 (50Mbps/64Kbps) simultaneous calls. Therefore, under ideal circumstances, the sum total of simultaneous calls from ER0, ER1 and ER2 should not exceed 781. Similar to that in Scenario 1, all calls were admitted and no packets were lost in the under-subscription scenario. Due to lack of space we only show results for the over-subscripted scenario. In this case, the mean arrival rate was fixed at 8 per second which results in an over-subscription level of approximately 150%.

D. Scenario 2: Results

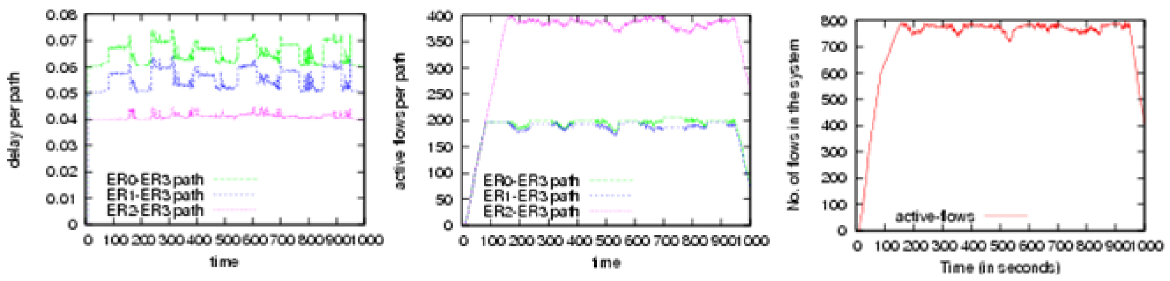


Figure 6. Scenario 2: Per path e2e delay, number of active calls and total active calls v/s time for the experiment with λ = 8

Fig. 6 shows the end to end delay and number of active calls in the system against time along each ingress egress path and also the total active calls (cumulated over all ingress egress paths) which simultaneously exist in the network at any point of time. Since the network is over-subscripted, some calls have to be rejected. The behaviour of the delay plot for each ingress egress path is along similar lines as that outlined in the previous scenario. In this case too, a few calls in excess of the network capacity are admitted into the system. The delay starts rising when these extra calls are admitted and stays high until the average delay reduces and becomes lesser than the Rejection Threshold (which happens in response to the number of active calls reducing below the maximum admissible limit). Observe from the active calls per path plot that close to 200 calls are admitted along the ER0-ER3 path, less than 200 calls are admitted along the

ER1-ER3 path and less than 400 calls are admitted along the ER2-ER3 path. About 3 to 5 calls each more than the capacity were admitted on each of the bottleneck links. All paths taken together, the number of calls admitted that exceed the capacity are about 9. Similar to that in the previous scenario, the buffer size was fixed at 60 packets. Due to the small buffer size and admission of extra calls, some packets were lost. The loss ratio observed due to admission of these excess calls was 0.16% which is very small. Overall, despite admitting a few extra calls, the proposed admission control algorithm is able to regulate the delay below the target delay and at the same time results in a low packet loss.

In summary, the proposed approach is capable of delivering on the stringent QoS requirements of voice traffic. It may not be able to provide as fine grained QoS as its hop by hop reservation based counterparts but its simple approach coupled with its lightweight implementation makes it a viable alternative for deploying voice over IP.

5. Conclusion

In this paper we proposed a scalable and distributed algorithm for call admission control of voice traffic and evaluated its performance in topologies with single as well as multiple bottleneck links. We also tested its performance under two different scenarios - one in which network services are under-subscribed and the other in which they are over-subscribed. Results from this study show that the proposed algorithm is capable of delivering the stringent QoS requirements of voice traffic by regulating the end to end delay below the target value of 150ms and at the same time achieving less than 1% packet loss. Even though it may not be able to provide as fine grained QoS as other existing approaches, its simplicity coupled with a lightweight implementation makes it a good candidate to be considered for deploying voice over IP. Despite its benefits, it has a limitation in that it operates only within a provider controlled domain. Extending this algorithm so that it operates across multiple provider domains will be the subject of our future investigation. We also intend to carry out a rigorous simulation based performance evaluation of the proposed algorithm in the presence of more complex topologies.

REFERENCES

1. HATTINGH, C. and SZIGETI, T., **End-to-End QoS Network Design: Quality of Service in LANs**, WANs, and VPNs. Cisco Press, 2004.
2. JAMIN, S., DANZIG, P., SHENKER, S., and ZHANG, L., **A Measurement-based Admission Control Algorithm for Integrated Service Packet Networks**, IEEE/ACM Transactions on Networking, Vol. 5, Feb. 1997.
3. KELLY, F., KEY, P. and. ZACHARY, S., **Distributed Admission Control**, IEEE Journal on Selected Areas in Communications, Vol. 18, pp. 2617–2628, Dec. 2000.
4. ELEK, V., KARLSSON, G., and RONNGREN, R., **Admission Control Based on End-to-end Measurements**, in Proceedings of the IEEE INFOCOM, Tel Aviv, Israel, pp. 623–630, Mar. 2000.
5. BRESLAU, L., KNIGHTLY, E., SHENKER, S., STOICA, I., and ZHANG, H., **Endpoint Admission Control: Architectural Issues and Performance**, in Proceedings of the ACM SIGCOMM, Stockholm, Sweden, pp. 57–69, Aug. 2000.
6. TANG, N., TSUI, S., and WANG, L., **A Survey of Admission Control Algorithms**, tech. rep., Computer Science Department, University of California Los Angeles (UCLA), Dec. 1998.
7. BELENKI, S., **An Enforced Inter-admission Delay Performance-driven Connection Admission Control Algorithm**, ACM SIGCOMM Computer Communication Review, vol. 32, pp. 31–41, Apr. 2002.
8. BHATNAGAR, S. and BADRINATH, B., **Distributed Admission Control to Support Guaranteed Services in Core-stateless Networks**, in Proceedings of the IEEE INFOCOM, Apr. 2003.
9. YUAN, P., SCHLEMBACH, J., SKOE, A., and KNIGHTLY, E., **Design and Implementation of Scalable Edge-based Admission Control**, Computer Networks (COMNET), Vol. 37, pp. 507–518, Nov. 2001.

10. GUERIN, R., AHMADI, H., and NAGHSHINEH, M., **Equivalent Capacity and its Application to Bandwidth Allocation in High-speed Networks**, IEEE Journal on Selected Areas in Communications, Vol. 9, pp. 968–981, Sept. 1991.

11. BRESLAU, L., JAMIN, S., and SHENKER, S., **Comments on the Performance of Measurement-based Admission Control Algorithms**, in Proceedings of the IEEE INFOCOM, pp. 1233–1242, 2000.

12. SCHEL´EN, O., **Quality of Service Agents in the Internet**. PhD thesis, Computer Science and Electrical Engineering, Sweden, Aug. 1998.

13. LAKKAKORPI, J., **A Flexible and Adaptive Connection Admission Control Framework for Differentiated Services Access Networks**, tech. rep., Department of Electrical and Communications Engineering, Helsinki University of Technology, Mar. 2004.

14. ZHANG, Z., DUAN, Z., GAO, L., and HOU, T., **Decoupling Qos Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services**, in Proceedings of ACM SIGCOMM, pp. 71–83, 2000.

15. KULKARNI, P., MCCLEAN, S., PARR, G., and BLACK, M., **Proactive Predictive Queue Management for Improved Qos in IP Networks**, in Proceedings of IEEE International Conference on Networking (ICN), 2006.

16. BRADEN, R., ZHANG, L., BERSON, S., HERZOG, S., and JAMIN, S., **RFC-2205 Resource Reservation Protocol**, tech. rep., IETF, Sept. 1997.

17. **ns-2 network simulator available at**. http://www.isi.edu/nsnam/ns/, Sept. 2006.

18. BRESLAU, L., JAMIN, S., and. SHENKER, S, **Measurement-based Admission Control: What is the Research Agenda**, in Proceedings of the International Workshop on Quality of Service (IWQoS) London, UK, pp. 3–5, May 1999.

19. **Desigining mpls networks for voice**. http://www.webtorials.com/main/eduweb /atm/tutorial/index.shtml, Sept. 2006.