A Lightweight, Scalable and Distributed Admission Control Algorithm for Voice Traffic

Parag Kulkarni∗†, Petre Dini1, Sally McClean∗, Gerard Parr∗ and Michaela Black∗

†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 similar to that 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 in excess of the network capacity 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 known as call admission control. This paper proposes a scalable and distributed call admission control algorithm that operates at the network edge and factors the local passive measurements into the admission decision. Performance evaluation of the proposed algorithm through ns2 simulations reveals that it is successful in detecting rate mismatches (input rate greater than output rate) and subsequently rejecting admission requests (as long as input rate is greater than output rate) thereby delivering on the QoS guarantees demanded by voice applications. Moreover, it is simple and lightweight from an implementation perspective.

I. 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 150ms, loss rate of 1% or less and an average jitter less than equal to 50ms (see [1] for details). If more connections† are admitted into the network than the network can cope with, the QoS of already admitted connections may be compromised (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 approaches 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

†In this paper, the terms ‘call’ and ‘connection’ are used interchangeably
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 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 ideally suited to a lightweight implementation.

II. 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 [15] 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. Typically, the Expedited Forwarding (EF) class within a DiffServ network is used to carry priority traffic such as voice traffic and hence it is assumed that each voice call is carried in the EF class with an end to end target delay of this class being 150ms$^2$. It is also assumed that the EF class is served with Strict Priority Queuing (SPQ)$^3$. 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

---

$^2$This value of target delay has been chosen in line with requirements of voice traffic as outlined in [1]

$^3$The main motivation for using SPQ is to isolate the effects of bursty traffic in other traffic classes on the delay experienced by packets in the EF class
network that pass through this exit point. This is key idea behind the algorithm which we will elaborate upon in the next subsection. It should be noted that since the algorithm is based on detecting rate mismatches at the egress through passive monitoring of delay, the need to estimate available bandwidth at each link is alleviated.

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 1. 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 1.

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.

III. 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 [16]. 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 egress node, single bottleneck link, homogeneous end to end delay

Experiments under this scenario were carried out on the topology shown in Fig 2. 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
Packet arrived at the Egress

\[ t_R \text{ – time when pkt was received, } t_{TS} \text{ – time as shown by timestamp field in pkt} \]

\[ \text{Inst}_e2e\text{Delay} = t_R - t_{TS} \]

\[ \text{Ingress}_id = \text{get ingress Id from the packet header} \]

Create state for this ingress in the EgressState linked list and update parameters for this ingress-egress pair.

\[ \text{ingressId} = \text{Ingress}_id \]

\[ \text{minDelay} = \text{inst}_e2e\text{Delay} \]

\[ \text{RejectionThreshold} = \text{minDelay} + (\text{minDelay} \times 0.1) \]

Add delay observation to the DelayHistory array

Get a pointer (Ptr) to the node which holds state for this ingress

\[ \text{Inst}_e2e\text{Delay} < \text{Ptr->minDelay} ? \]

YES

Update state for this Ingress

\[ \text{Ptr->minDelay} = \text{inst}_e2e\text{Delay} \]

\[ \text{Ptr->RejectionThreshold} = \text{minDelay} + (\text{minDelay} \times 0.1) \]

Compute average over the last ‘n’ delay values in

\[ \text{Ptr->DelayHistory} \]

YES

Queue Full ?

NO

Enque packet

Drop Packet

END

YES

Average <= \text{Ptr->RejectionThreshold}?

Set admit bit in reservation header of the packet to 1

Set admit bit in reservation header of the packet to 0

Fig. 1. Admission Control Logic

Fig. 2. Single bottleneck, homogeneous e2e delay

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. 3 shows the end to end delay and number of active calls in the system against time for both the aforementioned experiments with \(\lambda = 2\) and 4 respectively. As shown in figure, in the first case where \(\lambda = 2\) (Fig.3(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 (\(\lambda = 4\), Fig.3(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. The increase in delay (above

\(^4\)Assuming the bandwidth requirement of each call to be 64Kbps
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 3(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) in order to minimize the queuing delay of approx. 10ms. 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 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 signalling packets for these calls were received, the average delay was below the Rejection Threshold.

C. Scenario 2: Multiple Egress nodes, multiple bottleneck links, heterogeneous end to end delay

Fig 4 shows the topology used under this scenario. S0-D0, S1-D1 and S2-D2 are the source-destination pairs. Traffic from each of these sources encounters different fixed end to end delays while passing through the egress router ER0, ER1 and ER2 respectively on its way to the destinations (60ms, 50ms and 40ms respectively). The bottleneck link CR0-CR1 whose capacity is 40Mbps (maximum capacity in terms of number of simultaneous calls being 40Mbps/64Kbps = 625 calls) is shared by traffic from the two ingress routers IR0 and IR1 respectively. This link is the lowest capacity link on the end to end path for traffic coming from both these ingress nodes. Hence the sum of active calls from IR0 and IR1 traversing CR0-CR1 link should ideally not exceed 625. The CR1-CR2 link on the other hand is shared by traffic from IR0 and IR2. The capacity of this link is 25Mbps which approximately results in a call capacity of 390 (50Mbps/64Kbps) simultaneous calls. Therefore, ideally, the sum total of simultaneous calls from IR0 and IR2 passing through CR1-CR2 link should not exceed 390. 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-subscribed scenario. The mean arrival rate used in this experiment was fixed at 9 per second which results in an over-subscription level of approximately 150%.

D. Scenario 2: Results

Fig. 5 shows the end to end delay and number of active calls in the network along each ingress-egress path against time. Both these plots exhibit similar behaviour as that 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). Around 200 calls each are admitted along both the IR0-ER0 path and IR2-ER2 path and around 425 calls are admitted along the IR1-ER1 path. About 7 to 10 calls (on top of the capacity) were admitted on the CR0-CR1 bottleneck link and about 5 calls (on top of the capacity) were admitted.
along the CR1-CR2 bottleneck link. Similar to that in the previous scenario, due to a small buffer size (60 packets) and admission of surplus calls, some packets were lost. The loss ratio observed due to admission of these surplus calls was 0.431% 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.

IV. 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.

ACKNOWLEDGMENT

This work was done when the first author was an intern with Cisco Systems Inc. He would like to thank Cisco for providing this opportunity.

REFERENCES