Admission Control and Resource Management for a Bandwidth Broker (BB) in a DiffServ Domain

Jun Gao, Dimitrios Pendarakis

August 20, 1999

Abstract

In this project, we identified the major functionalities of a Bandwidth Broker (BB), a logical entity in a DiffServ domain for internal resource management and inter-domain bilateral Service Level Aggregation negotiation. We focused on the problem of admission control for aggregated flows across domains. In the DiffServ context, a BB has to decide whether to accpet an aggregated flow with incomplete information, because the destination domains of a request are often not specified and resource requirement for each individual flow within the request is also unknown. We designed a measurement-based probabilistic admission control algorithm for aggregated flows. The algorithm is based on the observation that for a link in a transit domain, the bandwidth requirement on this link is the sum of bandwidth contributed by all the possible ingress egress pairs that have this link on their paths. We estimate how an aggregated traffic from one ingress splits across the egress nodes by using a packet marking and egress counting mechanism. With measured traffic distribution ratios for each ingress node, the future bandwidth required on each link can be estimated and the probability that the requirement will be less than the link capacity can then be computed. The total probability that one aggregated request can succeed in the transit network can therefore be derived. We implemented Bandwidth Broker, a simplified protocol, the admission algorithm and the relevant mechanisms in ns-2. Simulation experiments were run to evaluate the system and the algorithm. Preliminary results validated the traffic matrix estimation mechanism and showed that a domain with a carefully selected admission control threshold can operate at a high utilization level while keeping request blocking rate and packet loss rate low.

1 Introduction

1.1 Background

Recent proposals suggest that a scalable model to provide QoS in the Internet is using DiffServ in backbone networks and stateful, per flow signaling protocols, e.g. RSVP, in stub networks. However since the Internet consists of multiple autonomous systems(ASs), in order to provide any kind of end-to-end quality of service each AS needs to establish bilateral agreements with its adjacent domains. These agreements will allow an AS to forward its aggregated traffic to destinations outside its domain and, conversely accept and carry traffic from other domains according to prescribed agreements. An entity called *Bandwidth Broker(BB)*, residing in each domain, has been proposed to manage intradomain resources and to negotiate bilateral agreements with neighboring Bandwidth Brokers.

Currently in the Internet agreements are configured statically and usually involve human intervention. However, the ability to efficiently guarantee QoS and to respond to dynamically changing traffic patterns requires that a scheme is devised to automate the negotiating process. In order to perform its duties a BB has to maintain information about the state of its domain, such as link utilization, packet loss rates and packet delays, and use inter-domain signaling to communicate with other BBs.

Bilateral agreement negotiations obviously involve pricing issues; a BB may charge differently for different services asked by neighboring BBs, it might base its pricing decisions on the state of its domain or it might even auction bandwidth to neighboring BBs. Similarly a BB requesting bandwidth should have the ability to talk to multiple BBs and choose the most favorable one that can satisfy its request.

1.2 Functions of a Bandwidth Broker

A BB must be able to generate and process resource requests to and from, respectively, neighboring BBs. In the case of an AS that contains end hosts, the BB should also be able to process intra-domain resource requests. In doing so, a BB must address the following problems:

1. **Admission control for aggregated flows**
   When a request for carrying given amount of traffic at given service level is received, *can* and *should* this request be accepted?

2. **Resource allocation and management**
   If the request is to be accepted, how should resources within the BB's domain be allocated and managed to satisfy the service request?

3. **Pricing**
   If the request can be accepted, how should a BB charge the requesting BB?

Additionally, a BB must decide when a request for service from neighboring BBs should be generated and for what amount of traffic. In general, an outgoing request is generated as a response to incoming requests from neighboring BBs, requests from stub networks attached to the BB's domain, or requests from hosts residing in the BB's domain.

2 Problem Description

Let us examine the example network shown in Fig. 1, which consists of four domains.

Consider BB2 receives a Resource Allocation Request(RAR) from its neighbor BB1. The request is in the form of flow aggregation (may or may not have the list of all destination domains) asking for bandwidth at a certain service level, e.g., premium service. The essential problem presented to BB2 is admission control for this aggregated flow. This is considerably difficult comparing to end-to-end flow admission control, basically because BB2 does not have the information of how this flow distributes itself among the possible downstream domains.

If a request is admitted by the admission control algorithm, the BB can then trigger other intra-domain mechanisms in the system to allocate and manage the resources inside its domain. For example, BB may use RSVP as the signalling protocol to make reservations from ingress to egress nodes for the admitted flow, or BB may reconfigure the edge devices to accomodate this flow if DiffServ components were implemented in this domain.

In this report, we present an admission control algorithm for inter-domain aggregated flows. The algorithm is based on a measurement-based traffic matrix estimation mechanism. We implemented this algorithm in a simulation environment, ns-2, along with a simplified version of inter-domain BB communication protocol, so that we have a running system. In Section 3, we describe the admission

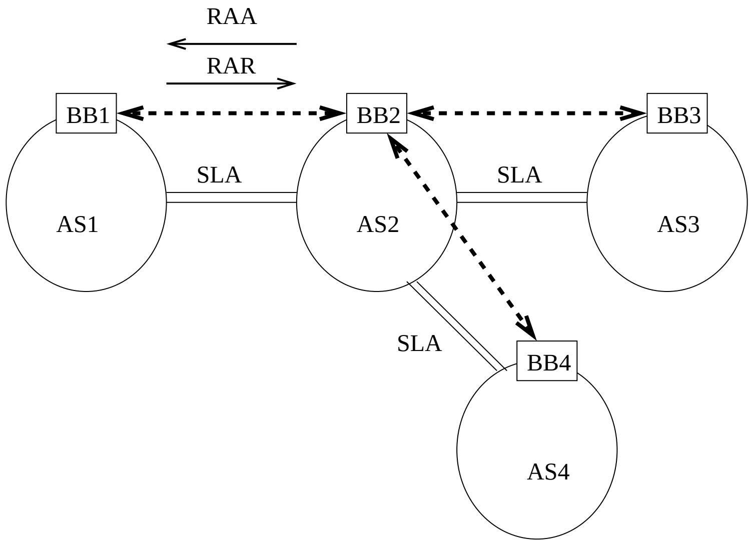

Figure 1: Inter-domain BB communication.

control algorithm. The details of the implementation is presented in Section 4. To evaluate our algorithm and the system, we run a series of simulation experiments, and the results are shown in Section 6. We conclude the report in Section 7 with conclusions and future work.

3 Admission Control for Aggregated Flows

Admission control for an end-to-end flow is fairly staightforward. With destination known, the route from the source to the destination can be determined by following the routing table, a signalling protocol can just go through the routers on this route to make sure each link has enough resource to accomodate this flow.

However, the picture gets complicated in the case of resource management across mulitple Autonomous Systems. The inter-domain resource request is in the form of flow aggregation, and the possible destination domains are often not known even to the requesting BB. Given incomplete information, it is difficult for the requested BB to make a proper admission control decision.

If use statically configured SLA, when receiving a request, a BB can simply consult with a (policy server) database to see whether the current SLA with the requesting domain can satisfy this new request. Presumably the pre-negotiated SLA may be based on some heuritics of how the request from one domain may split across egress nodes, but since this scheme does not incorporate any of the current status of the network, it may make wrong decisions. An extreme case would be the new request's data traffic concentrate on one or only a few egress nodes, congestion will be formed close to those nodes if the request is admitted.

It would help the BB to make a more intelligent decision if the destination domains from one domain's requests can be approximated and the portion of traffic that goes to each destination domain can be estimated. A BB can then decide to accept or reject a new request based on the temporal locality property of the system.

In this section we describe a measurement-based probablistic admission control algorithm for aggregated flows.

3.1 Traffic matrix estimation

Consider an AS that has n edge routers, i.e., connecting to n other domains. We have the following equation to express the traffic that passes this domain:

$$\begin{bmatrix} e_1 \\ e_2 \\ . \\ . \\ e_n \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} & \ldots & F_{1n} \\ F_{21} & F_{22} & \ldots & F_{2n} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ F_{n1} & F_{n2} & \ldots & F_{nn} \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ . \\ . \\ i_n \end{bmatrix},$$

or,

$$\vec{E} = F \cdot \vec{I},$$

where, i_i is the total bandwidth request received from BB_i, F_{ij} is the fraction of incoming traffic from ingress point j and exiting at egress point i, and e_i is the bandwidth to be used at egress router i.

The traffic matrix F expresses how traffic entering a domain at different ingress points is distributed across different destinations. Specifically, element F_{ij} of column j of F represents the percentage of traffic entering at ingress point j that is destined to egress point i. Knowledge of the matrix F, coupled with topology and routing information allows a bandwidth broker to construct an accurate picture of utilization of all domain links.

In the absence of egress AS specification in a bandwidth request from a neighboring BB, the matrix F can be estimated in a number of ways, which can be broadly categorized into two groups. The first relies on adhoc assumptions, based on topology, routing and link capacity knowledge, and the second relies on measurements. Next we propose a measurement-based traffic matrix estimation scheme followed by a comparison of our scheme with some other schemes.

3.1.1 A measurement-based scheme

The set of egress points and the associated required bandwidth can be estimated using the following algorithm.

In this scheme the ingress node marks packets with the ID of the ingress node (equivalently that of the adjacent AS) and egress nodes, upon receiving marked packets, accumulate a count of packets received per ingress node. BB can retrieve these statistics by periodically querying edge nodes, and can then construct the matrix F. For example, suppose at ingress point 1, 100 packets were marked, and 50 showed up at egress 2, 30 showed up at egress 3, 20 showed up at egress 4, then $f_{21} = 0.5, f_{31} = 0.3, f_{41} = 0.2$.

To achieve the marking, the ingress node could add an IP option to the packet which is processed only at the egress node. To reduce the additional processing, marking could be selectively, according to a sampling technique. For example, instead of marking every packet, we may mark one packet for every 10 packets.

One potential problem of this egress counting scheme lies in that if packets get dropped inside the transit network, the estimation will not be accurate. This can be fixed by recomputing a corrected fraction by combining loss rate information on each link from ingress point to egress point with the measured fraction. This fix requires BB to know the path from each ingress to each egress point. Each router in this network also needs to be able to collect and report packet loss information on the links that it connects to.

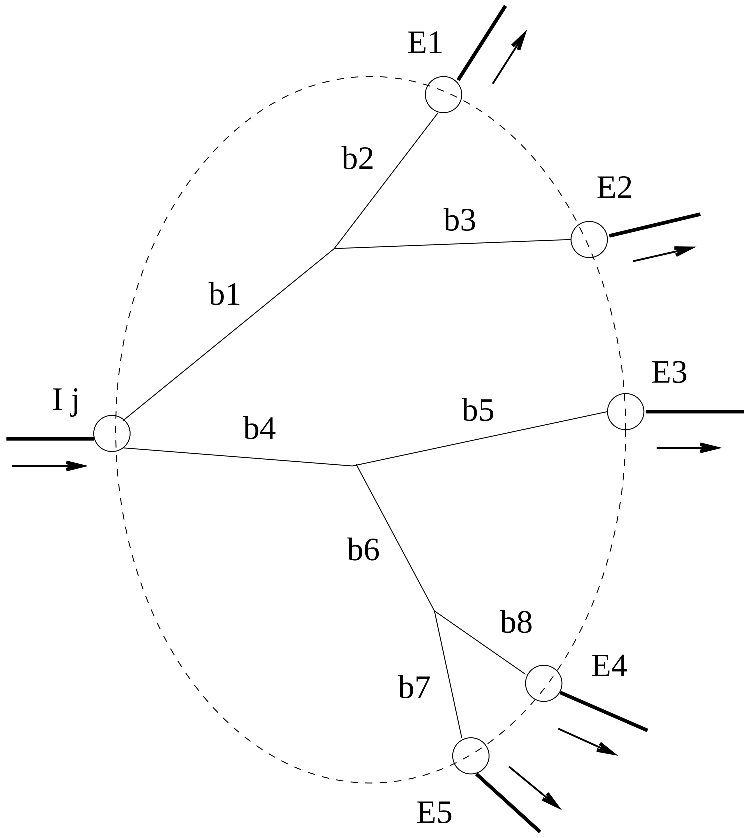

Figure 2: Estimation of matrix F based on ingress routed tree.

Continue using the example shown in figure 2, suppose the loss rates on the two links from I_j to E_2 are l_1 and l_2 respectively, and the measured fraction from I_j to E_2 is f_{2j}, so the corrected fraction is $f'_{2j} = \frac{f_{2j}}{(1-l_1)\cdot(1-l_2)}$.

Another disadvantage of this scheme is that it requires modification of the packet, which might be undesirable from a security and router processing point of view. An alternative to marking packets using IP option is to send a control packet at the ingress node with the same header as what is in the data packets whenever a marking action should be taken. This way no modification to the original data packet is needed.

3.1.2 Comparison with other schemes

Next, we examine a few other potential schemes for estimating the matrix F. We first look at two schemes that are based on topology considerations.

- The percentage of traffic entering an ingress point, I_j, that is destined to each possible egress point is proportional to the capacity of the egress links. Refering to figure 2, traffic entering through I_j is destined to egress points $E_1, E_2, \ldots, E_5$ with probability proportional to the capacity of the egress links.

- At each branching point of the ingress routed tree, traffic is split with some probability across the different links. This probability could either be fixed for all links or proportional to the link capacity. For example, in figure 2, the percentage of traffic entering at I_j which is destined to E_2 is $1/2 \times 1/2 = 1/4$, if branching probabilities are equal at each splitting point, or $\frac{b_1}{b_1+b_4} \cdot \frac{b_3}{b_3+b_2}$, if branching probabilities are proportional to link capacities.

These schemes can be used as a coarse approximation when there are lack of support from the edge nodes of the network. They are also useful as the initial intelligent estimate of the traffic matrix when the system starts for the measurement-based schemes.

Another measurement-based mechanism was proposed in [2TIER]. Count packets arriving at the ingress node and tabulate them according to the next hop AS based the destination IP address. The counting mechanism relies on the ingress node maintaining BGP routing information and thus using a function `egress(dest(i))` function to determine the egress router, `egress` for packet i with destination address `dest(i)`. Using this mechanism the ingress node approximates a column of the matrix F. This information can be reported to the BB and used for admission control and resource reservation purposes. The clear disadvantage of this approach is that it increases packet forwarding complexity and it relies on interfacing with BGP tables.

The benefits of our egress counting mechanism are that it requires less changes in the forwarding path and does not need to work with BGP. Reaggregated bandwidth requirements are immediately known at the egress points. This will help to determine whether new RARs should be triggered to downstream BBs.

3.2 Admission Control algorithm

Suppose transit domain AS0 has n edge nodes, i.e., AS0 connects to n other ASs, AS1 to ASn. Bandwidth broker, BB0, of AS0 maintains the measured matrix, F, input vector, $\vec{I}$, and output vector $\vec{E}$.

We make the assumption that the fractions, F_{ij}, in matrix F are independent Gaussian variables. [1] This is to say that for any ingress, egress pair (p, q), the fraction follows a Normal distribution, $N(f_{qp}, \sigma_{qp})$, where f_{qp} is the average measured fraction and σ_{qp} is the measured standard deviation. Take one arbitrary link, l_a, which has a capacity, C_a, we maintain the set of all ingress egress pairs that has this link on its route. The bandwidth needed on this link is then,

$$B = \sum_{all\ pairs\ (p,q)} i_p \cdot F_{qp}$$

Since this is a linear combination of independent Gaussian variables, B is also a Gaussian variable with a distribution $N(\mu, \sigma)$, where

$$\mu = \sum_{all\ pairs\ (p,q)} i_p \cdot f_{qp},$$

and

$$\sigma = \sqrt{\sum_{all\ pairs\ (p,q)} i_p^2 \cdot \sigma_{qp}^2}.$$

Suppose a new request comes in from ASk asking for additional r_k unit of bandwidth, and link l_a is on the route from k to one or more egress nodes. i_k is temporarily updated to $i_k = i_k + r_k$. μ and σ of this link are recomputed using the above formulars. And this new μ is our predicted average bandwidth requirement. However the real bandwidth usage, B, may deviate from this average, we want to compute the probablity that this future traffic will not exceed the link capacity, i.e.,

[1] the sum of one column is 1

$$P(B < C_a).$$

It can be computed easily by converting B to a standard normal distribution $N(0,1)$.
Let

$$z = \frac{B - \mu}{\sigma},$$

and now

$$\begin{aligned} P(B < C_a) &= P(\tfrac{B-\mu}{\sigma} < \tfrac{C_a-\mu}{\sigma}) \\ &= P(z < z_0) \\ &= \Phi(z_0) \end{aligned}$$

For ingress node, k, BB maintains the set of all links, $L_k = \{l_{k1}, l_{k2}, ..., l_{km}\}$, that are on the path from this node to all the possible egress nodes. To compute the probablity that this new request will succeed, the admission control algorithm has to compute the probablity that it will succeed on all the links in L_k using the above method. It is reasonable to assume that the event that it will succeed on one link is independent of the event that it will succeed on another link, thus the probability that this new request will succeed in the network is:

$$P = \prod_{t=1}^{m} P_{l_{kt}}.$$

In the implementation, if the predicted B is greater than C_a, we do not do all the probablity computation and simply reject the request. A BB selects an admission control threshold, T, as the criteria to accept a request. BB accepts a request only when $P > T$. BB can dynamically adjust the threshold to optimize the its network's status.

3.3 Intra-domain resource management and Resource Request propagation

Once the traffic matrix is estimated, in addition to being utilized for admission control purpose, it can be used for intra-domain resource management. For example, if RSVP is used for intra-domain signalling, once the bandwidth requirements to the egress points from an aggregated flow from one ingress point are estimated, reservations can be made within this domain from the ingress point to egress point(s).

Furthermore, when a request is accepted, a BB can immediately recompute the output vector $\vec{E}$, which represents the bandwidth requirements for this domain to its downstream neighboring domains. For example,

$$e_i = \sum_{j=1}^{n} f_{ij} \cdot i_j,$$

is the new bandwidth requirement to domain ASi. The BB can then decide whether to trigger a new RAR to downstream BBi based on the newly computed e_i and the existing bilateral agreement with the downstream BB(s).

4 Implementation

Bandwidth broker, the interdomain BB communication protocol, the probabilistic admission control algorithm, the packet marking and counting traffic matrix estimation mechanism are implemented in a simulation environment, ns-2.

4.1 Bandwidth Broker Agent

An agent in ns-2 represents an endpoint where network layer packets are constructed or consumed, and it is used for implementation of protocols at various layers.

Bandwidth broker is implemented as a special purpose agent running on the administrative node within each AS domain.

4.1.1 BB Interdomain protocol

We have implemented a simple interdomain protocol for a BB to communicate with adjacent BB. Currently, three interdomain messages exist: BB_MSG_REQUEST, BB_MSG_ANSWER, and BB_MSG_CANCEL. A BB agent sends a message with type BB_MSG_REQUEST to request a specified amount bandwidth from its neighbor BB agent. When a BB receives such a message, it will execute its admission control algorithm as explained in the previous section to decide whether to accept this request or not. The BB then sends a message with type BB_MSG_ANSWER to reply to the requesting BB. In the header of this message, BB sets the field answer to BB_REQ_ACCEPT if the request is accepted, or BB_REQ_REJECT if otherwise. The way for a BB to cancel a request is to send a message with type BB_MSG_CANCEL. When it is being received by a BB agent, the specified amount of bandwidth will be subtracted from the ingress vector.

4.1.2 BB Intra-domain management

To update matrix F and compute the distribution of each element in F, a BB agent needs to query edge nodes periodically to retrieve the number of bytes have passed this node. For this purpose, on each edge node, we put an edge BB agent which needs only to understand two types of BB messages, BB_MSG_QUERY, and BB_MSG_REPLY. Domain BB agent periodically sends messages with type BB_MSG_QUERY to edge BB agents and when these messages are received, edge BB agent retrieves the current bytes info from their data collecting component (the classifier) and attach this info in the data area of the replying message which has a type of BB_MSG_REPLY.

Master BB needs to collect congestion status on each link to correct matrix F in the case of packets dropped inside the network. This requires we run yet another agent on each intermediate node from an ingress to an egress. This agent, QueryAgent is responsible for retrieving packet loss information on each link that the node connects to as a response to the received BB_MSG_QUERY_QUEUE message from domain BB agent. The reply message to the domain BB has a type of BB_MSG_REPLY_QUEUE with the queuing info in the data area.

4.2 Supporting mechanisms

Several mechannisms are required for the BB system to work, namely packet marking, packet counting and link monitoring. For a detailed description of these components, please refer to the Appendix section.

4.3 The System and simulation script

ns-2 uses OTcl as the front end to configure a simulated network and to define the commmands and actions used in the simulation. In our simulation, we follow the usual steps of creating nodes, creating links and setting up various agents on different nodes. Specifically, the links between domains are edge links, and the marking object(stamper) on these links will mark packets with their edge id. For a domain, two kinds of nodes can be created, edge nodes which are specified with an edge id, and internal nodes which have edge id 0. Edge nodes have the ability to count bytes per ingress node.

For each simulation, a domain BB agent is configured on an administrative node, an edge BB agent is set up on each edge node, and query agents are set up on the intermediate nodes. BB agents of adjacent domains can send requests to each other and if a request is accepted, traffic from that domain will then follow. The requested domain BB does admission control based on its online traffic matrix estimation.

5 Simulation Experiments

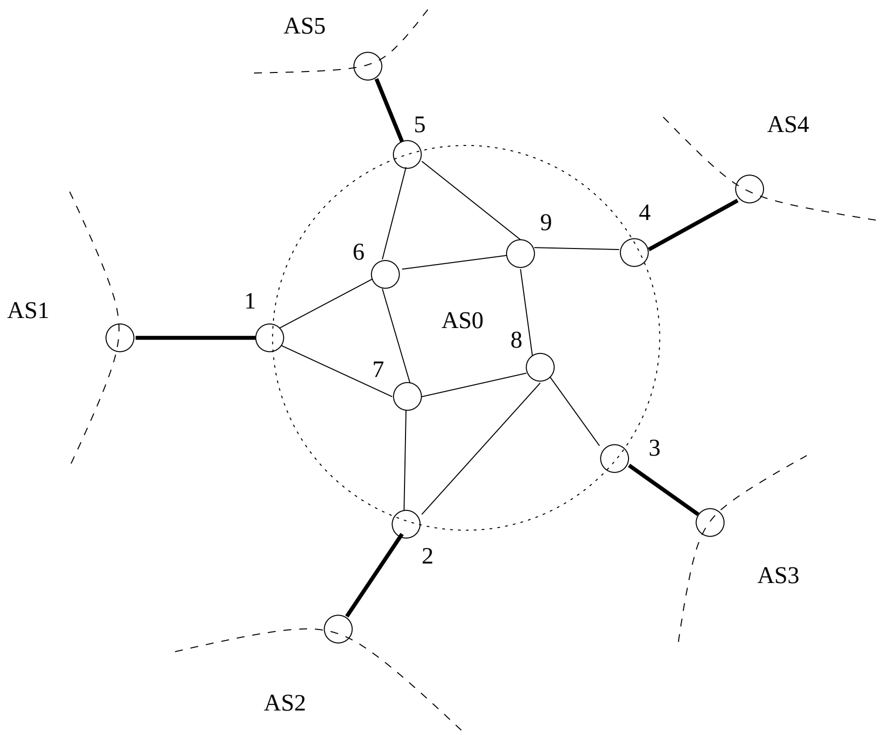

Figure 3: Topology used in simulation experiments

To validate the measurement-based traffic matrix estimation algorithm and see how effective the probabilistic admission control algorithm is, after the Bandwidth Broker protocol and the relevant mechanisms were implemented in ns-2, we conducted a series of simulation experiments.

The topology shown in Fig.3 was used for this section's experiments. There are five stub domains, AS1 to AS5, and each domain is represented by one node. These domains connect to a common transit domain, AS0. In AS0, node 1 through 5 are edge nodes, and node 6 through 9 are internal nodes. The route from one edge node to another edge node is generated based on shortest path algorithm. For example, the route from 1 to 2, is 1-7-2, and from 1 to 3 is 1-7-8-3, etc.

The links between two domains are 1Mbps duplex links, and the links inside domain AS0 are 0.5Mbps duplex links.

5.1 Validation of the traffic matrix estimation algorithm

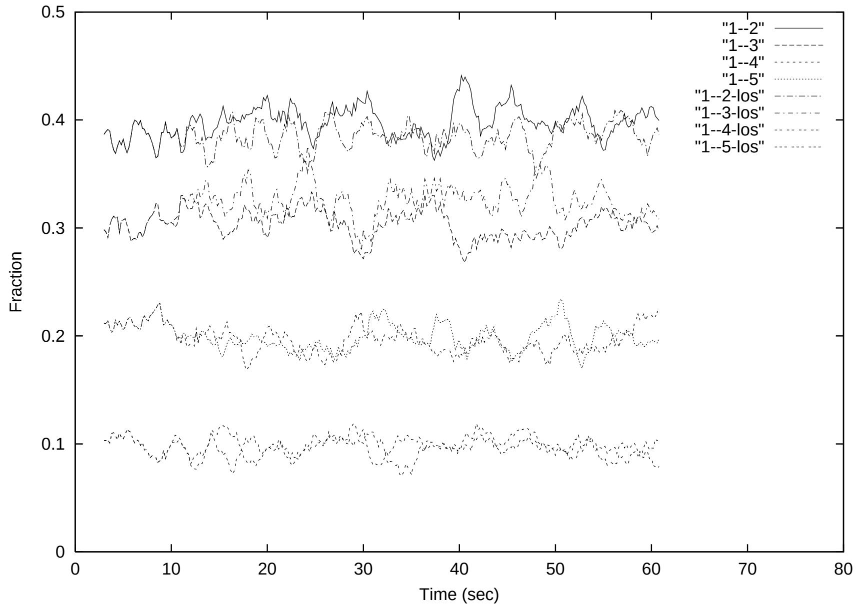

Figure 4: Traffic matrix test

In this section, we want to validate the measurement-based traffic matrix estimation algorithm, i.e., marking packets with an edge id at each ingress edge node, and counting packets according to the marked edge id at egress nodes to determine how an aggregated flow split accross the possible egress nodes.

In the first experiment performed, we generated an aggregated CBR traffic with a rate of 0.5Mbps in AS1 and it splits to AS2, 3, 4 and 5 with fractions 0.4, 0.3, 0.1 and 0.2. Each fraction follows a Gaussian distribution with the forementioned average and the standard deviations are 0.05. We measure the fractions every 0.2 seconds and each data point is the moving average of the past 10 measurements. The first four curves shown in Fig.4 are the measured fractions. As can be seen, the measurements follow the real split nicely. This validates the marking-counting mechanism.

In this experiment, there were no packets lost in the transit domain. One concern was that if packets get dropped inside the transit domain, the measurement maybe inaccurate. To handle this in our implementation, we also monitor the loss rate on each link, and we correct the fractions taking into account the loss rates from ingress to egress points. To see how well this works, in the second experiment, we add another aggregated flow with a rate 0.8Mbps from AS2 at time 10. This flow has an average split of 0.3125 to AS1, 0.5 to AS3, 0.0625 to AS4 and 0.125 to AS5. It causes congestion on link 7-3. The estimated split of flow from ingress node 1 to the other four egress points after correction with loss rate are shown as the last four curves in Fig.4. They are all around their expected values. However the difference from the case that no packet losses are clear particularly for traffic from node 1 to node 3 and from node 1 to node 2 because these two are directly affected by packet losses on link 7-3.

One optimization of the marking mechanism is that instead of marking every packet at the ingress node, we may achieve the estimation by only marking a portion of the packets that are passing through the ingress node. Fig.5 shows the comparison of the measured fraction from AS1 to AS2 under different marking frequency. In these experiments, all other parameters are kept the same as experiment 1. The annotation "n pkts" means for every n packets mark one packet. As

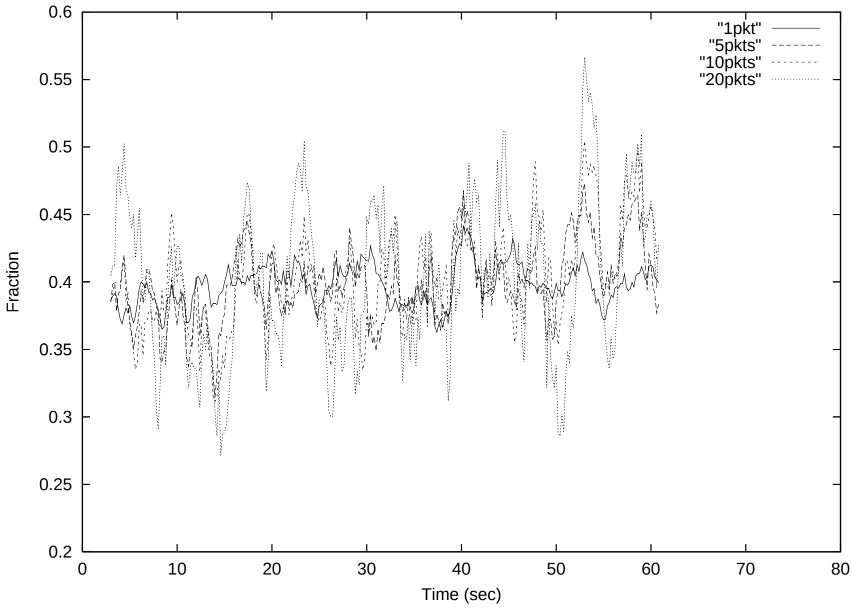

Figure 5: Comparison of traffic matrix measurement under different packet marking frequency

expected, with less packets marked, the measurement varies more dramatically. The mean and standard deviation were computed and shown in Table 1. The mean value accross the table are about the same, and in the case that every packet is marked, it has the smallest standard deviation. However to trade accuracy of the measurement with performance, marking less packets (e.g., every 5 or 10), also gave reasonable accuracy.

packets marking frequency	avg. fraction	standard deviation
1	0.398	0.0148
5	0.399	0.0277
10	0.402	0.0372
20	0.403	0.0653

5.2 Effectiveness of admission control algorithm

To evaluate how effective the admission control algorithm is, we run the simulation as a dynamic system, i.e., BB at one domain dynamically generates Resource Allocation Requests (RAR) to another domain and the requested domain BB does admission control to decide whether to accept or reject this request. Once a RAR is accepted, requesting domain BB will then generate real aggregated data traffic. We are interested in the following metrics, namely request(call) blocking rate, maximum link utilization, and the loss rate of the network. A good admission control algorithm should keep the call blocking rate relatively low while the link utilization high and loss rate low.

In the first set of experiments, AS1 and AS2 generate requests to AS0. All the requests from AS1 ask for the same amount of bandwidth, 0.04Mbps and have same egress split fractions (0.4, 0.3, 0.1, 0.2). Each request from AS2 also request 0.04Mbps bandwidth, but has a split fractions (0.25, 0.25, 0.25, 0.25). As a base case, the requests come in sequentially every 1 second and if they are admitted, their traffic last 20.5 seconds. AS1 sends its first request at time 1 and AS2 starts its first request at time 2. We run the simulation for about 70 seconds and the results are presented in Fig.6 to Fig.8.

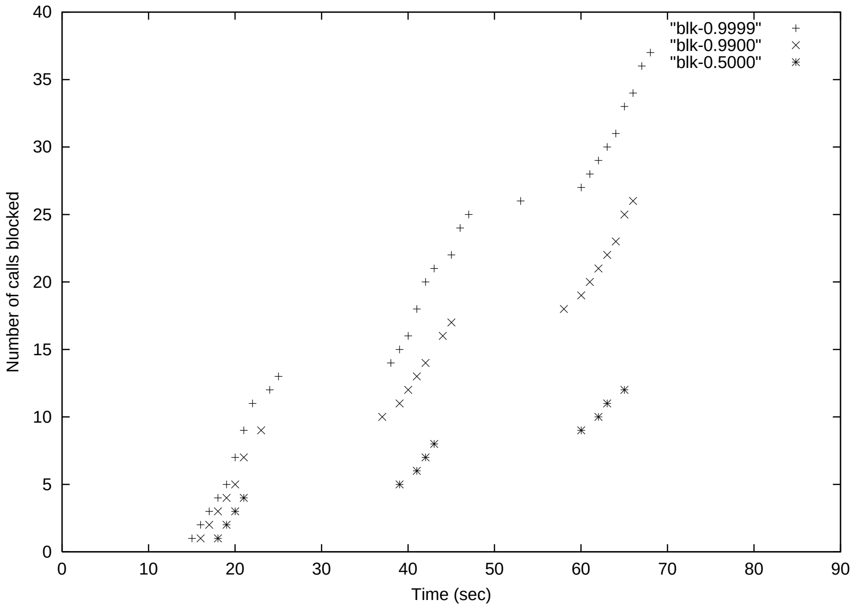

Figure 6: Cumulative number of requests blocked under different admission control thresholds for calls arriving with constant rate

Fig.6 shows the cumulative number of requests that are rejected under three different admission control thresholds. We specify an admission control threshold for each simulation run. The admission control algorithm admits a request only if the predicted probability of this request will succeed on all the links that are involved exceeds the threshold. Three different values, 0.9999, 0.9900, and 0.5000, are used as thresholds. As expected, when the threshold is set higher, more requests are rejected.

Fig.7 shows the maximum bandwidth utilization percentage on the most congested link when a new request comes in. The network gets saturated at around time 15 for all three cases. This is reflected by the fact that the three curves overlap with one another before that. After that, with a higher threshold, e.g., 0.9999, more requests are rejected and the network is kept in a lower utilization state. On the other hand, for a lower threshold, e.g., 0.5000, more requests are accepted and the link utilization of the network is higher.

In Fig.8 we show the loss rate within the network for the three cases. As the link utilization is higher, losses occur more often when more requests are accepted with less certainty of success.

Table 2 summarizes the result of these experiments.

a.c. threshold	overall blocking rate (%)	avg. max link util.(%)	number of links with losses
0.5000	8.76	88.3	32
0.9900	19.0	80.5	23
0.9999	27.0	73.7	7

A second set of experiments were carried out to simulate a real network situation. AS1, AS2 and AS5 are generating requests and data traffic. For each domain, the time interval between two consecutive requests follows an exponential distribution with an average of 1 second. The duration of each request also follows an exponential distribution but with an average of 20 seconds. By using different random seeds, the exponential distributions for one domain are different from the distribution of other domains.

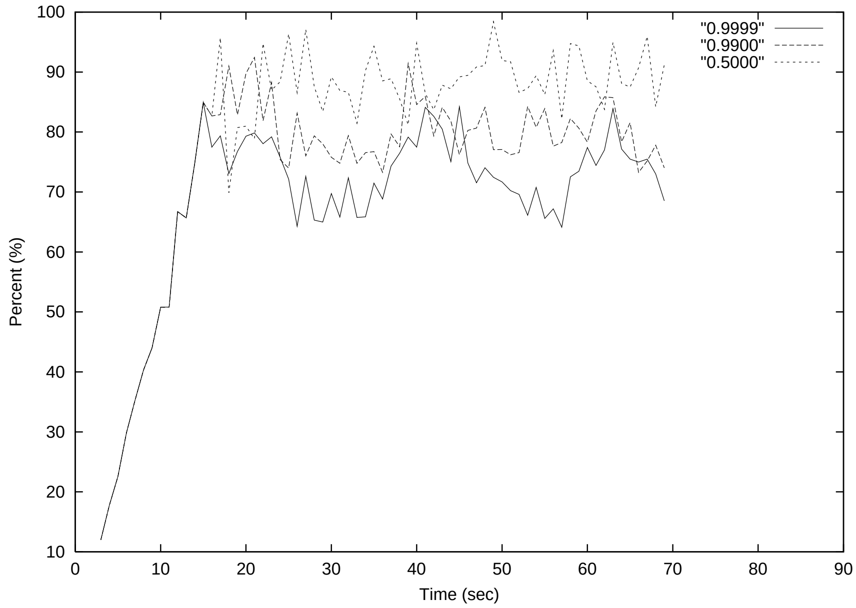

Figure 7: Maximum link utilization under different admission control thresholds for calls arriving with constant rate

Four thresholds, 0.5000, 0.9000, 0.9900 and 0.9999, were used in these experiments. Similar results were obtained and are shown in Fig.9 to Fig.11 and Table 3. As seen in Fig.10, when the threshold is too low, 0.5000, link utilization often reaches 100condition with lots of losses while the average link utilization is actually not the highest. When the admission control threshold is set to be 0.9000, the blocking rate is a little higher than when the threshold is 0.5000, but it achieves a higher link utilization with less losses and the system is more stable. However if the threshold is set too high, e.g., 0.9999, more than 50of the calls are blocked and the network is under utilized. This set of experiments indicates that to maximize the network utilization and minimize the call blocking rate, the admission control threshold should be chosen carefully by the BB based on the dynamics and characteristics of the traffic.

a.c. threshold	overall blocking rate (%)	avg. max link util.(%)
0.5000	34.5	85.03
0.9000	39.2	88.03
0.9900	45.6	73.37
0.9999	51.6	67.66

6 Summary and Future work

6.1 Summary

A scalable, two-tier resource management model to provide QoS for the Internet has been recently proposed in the literature. There are two hierachies in this model which are inspired by the current Internet routing. For inter-domain resource management, to carry traffic from one Internet AS to another with certain guarantees, bilateral Service Level Agreement needs to be set up between two adjacent domains. Within one stub domain, since there are less number of hosts and flows, stateful, per-flow signalling protocols like RSVP can be used to reserve resources. A logical entity, Bandwidth Broker(BB) has been suggested for each AS domain to both manage intra-domain resources and

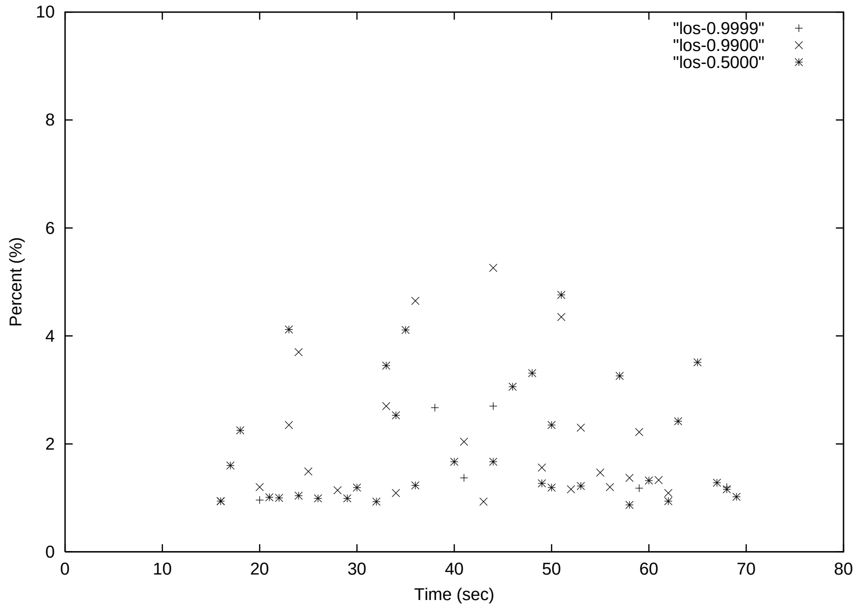

Figure 8: Maximum loss rate under different admission control thresholds for calls arriving with constant rate

to negotiate bilateral aggreements with neighboring BBs. In this project, we identified the major functionalities of a BB and focused on the admission control problem, which is the central problem a BB faces. The difficulty lies in that in the DiffServ context, for aggregated requests between domains without an explicit specification of destination domains and resource requirement for each individual flow, how to perform admission control for a BB. We designed a measurement-based admission control algorithm for aggregated flows across domains. The algorithm is based on the observation that for a link in a transit domain, the bandwidth requirement on this link is the sum of bandwidth contributed by all the possible ingress egress pairs that have this link on their paths. We estimated how the aggregated traffic from one ingress split across the egress nodes by using a packet marking and egress counting mechanism. With measured traffic distribution ratio for each ingress node, the future bandwidth required on each link can be estimated and the probability that the requirement is going to be less than the link capacity can then be computed. The total probability that the aggregated request can succeed in the transit network can therefore be derived. We implemented Bandwidth Broker, a simplified protocol, the admission algorithm and the relevant mechanisms in ns-2. Simulation experiments were run to evaluate the system and the algorithm. Preliminary results validated the traffic matrix estimation mechanism and showed that a domain with a carefully selected admission control threshold can operate at a high utilization level while keeping request blocking rate and packet loss rate low.

6.2 Future work

6.2.1 On-going work

Currently we are working on to make the system work for requests cross multiple AS domains, i.e., when a request is admitted by one domain, one or more new requests may be prompted to downstream domains as the response to the initial request.

We are also expanding our implementation to handle multiple traffic classes for a DiffServ

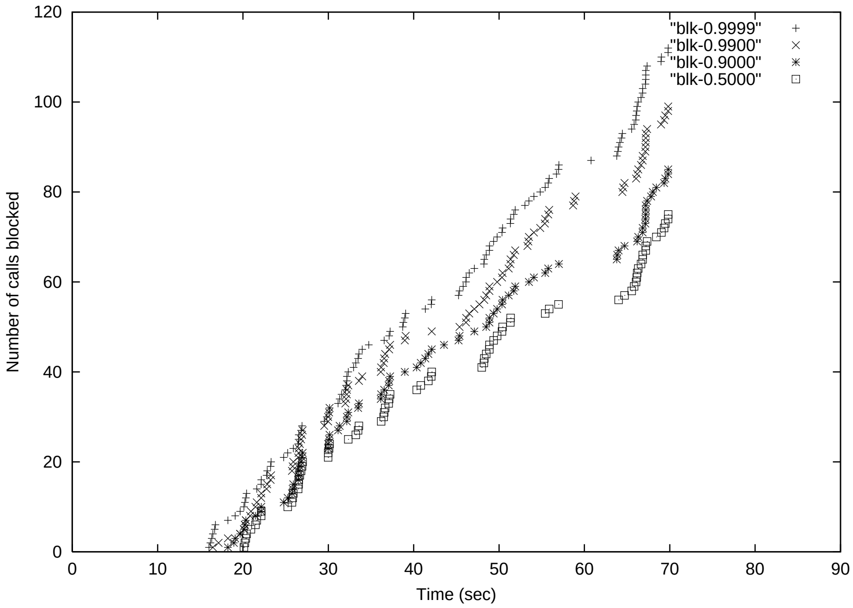

Figure 9: Cumulative number of requests blocked under different admission control thresholds for calls arriving with exponential rate

domain. The current implemenation considers only one class, the premium service.

We are planning to have an implementation in a real network and to analyze how the system would work in a network with real data traffic.

6.2.2 Pricing

This is a rather hard problem. Part of it is that it is unclear what is the utility function that we want to maximize. This has not been our focus so far, but the following are some ideas we had discussed.

Seller

If a request is accepted, what should be the price that is put on this service?

1. One simple scheme.

$$price = \sum \frac{congestion\ status\ of\ a\ link}{link's\ distance\ to\ the\ ingress\ edge\ router}$$

   i.e., if a link is highly congested, charge more, or if it is more closer to the ingress, (more critical) charge more. Congestion status can be represented by delay, or inverse of the percentage of available bandwidth on a link.

2. A slightly refined scheme. Accounting and prediction based on history.

   Maintain a profile for BB1's traffic. Record the possible destinations it often wants to go to and the bandwidth used (The first column of matrix F.). Combine this info with the known bw/delay, and put different weight factors on different links and egress routers.

Buyer

Now from requesting BB point of view as a bandwidth buyer. There are at least two schemes here.

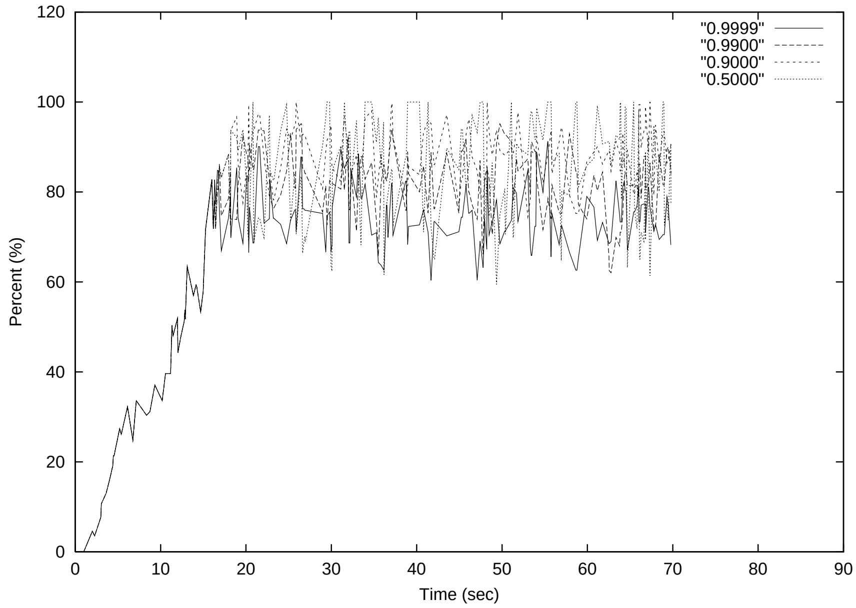

Figure 10: Maximum link utilization under different admission control thresholds for calls arriving with exponential rate

1. Buyer sends a request to multiple sellers and each seller returns with a price. For example, BB1 sends a RAR to each of its available adjacent BB. Among the "yes" answers, take the one that is most favorable to it, e.g., the cheapest one in price. In this case, it is the seller(s) decides the price.

2. Seller is selling a certain amount of bandwidth to multiple buyers. This is an auction scheme. Buyers make offers to the seller and the seller chooses one buyer to the best of its interest. In this case, it is the buyer who decides the price.

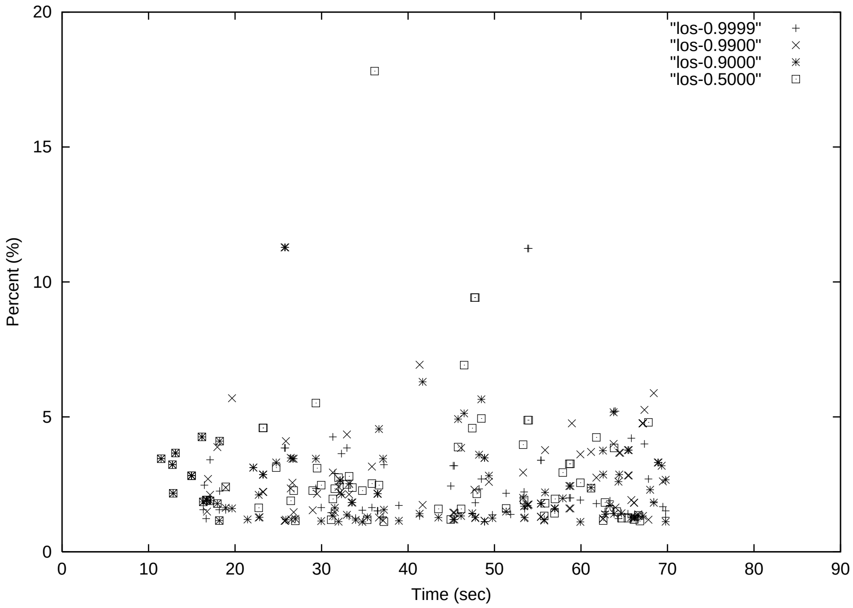

Figure 11: Maximum loss rate under different admission control thresholds for calls arriving with exponential rate

References

[1] K. Nichols, V.Jacobson and L.Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, ftp://ftp.ee.lbl.gov/papers/dsarch.pdf, November 1997.

[2] F. Reichmeyer, L.Ong, A. Terszis, L. Zhang and R. Yavatkar, *A Two-Tier Resource Management Model for Differentiated Services Networks*, IETF draft, November, 1998.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, *An Architecture for Differentiated Services*, RFC 2475, December 1998.

[4] N. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan and J. E. van der Merwe, *A Flexible Model for Resource Management in Virtual Private Networks*, Proceedings of Sigcomm'99, Boston, 1999.

[5] R. Neilson, J. Wheeler, F. Reichmeyer and S. Hares, *A Discussion of bandwidth Broker Requirements for Internet2 Qbone Deployment*, version 0.6, May, 1999.

[6] B. Teitelbaum, Draft QBone Architecture, Internet2 QoS working group draft, May, 1999.

[7] R. Yavatkar, D. Pendarakis, and R. Guerin, *A Framework for Policy-based Admission Control*, April, 1999.

[8] N. Semret, R. Liao, A. Campbell and A. Lazar, *Market Pricing of Differentiated Internet Services*, Proceedings of IWQOS'99, London, 1999.

[9] E. Knightly and N. Shroff, *Admission Control for Statistical QoS: Theory and Practice*, IEEE network, 13(2):20-29, March, 1999.

[10] S. Jamin, P.B. Danzig, S. J. Shenker, and L. Zhang, *A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks*, Proceedings of SigmComm'95, pp 2-13, 1995.

A Implementation

Bandwidth broker, the interdomain BB communication protocol and the relevant mechanisms are implemented in a simulation environment, ns-2.

A.1 Module description

A.1.1 Packet marking

`Class Stamper` provides the packet marking mechanism. It is a class derived from `Class Connector`. When configure an edge link, `Stamper` is added to the link as a component, which is similar to `TTLChecker` component. Each `Stamper` instance's sole member `edge_id_` is uniquely initialized with the edge link's edge id at network configuration time. The marking itself is rather simple, for each packet that passes the `Stamper`, `Stamper` updates the IP option field, `ip_opt` with its `edge_id_`.

```
        void recv(Packet* p, Handler* h) {
                hdr_ip* iph = hdr_ip::access(p);

                iph->ip_opt() = edge_id_;

                send(p, h);
        }
```

A.1.2 Packet counting

In ns, the function of a node receiving a packet and forwarding it to the next hop if it is not the intended receiver is provided by a `Classifier` object. For the purpose of counting packets at egress points, we derive a new class, `EdgeClassifier` from the basic classifier class `AddressClassifier`. Besides the usual task of examining the destination and forwarding, the `EdgeClassifier` object looks at the packet's `ip_opt` and increments the appropriate counter in its internal data structure `collector_`.

```
int EdgeClassifier::classify(Packet *const p) {

        hdr_ip* iph = hdr_ip::access(p);
        hdr_cmn* cmnh = hdr_cmn::access(p);

        // temporary implementation of a collecting table

        // 777 is the default value, means a control packet
        if(iph->ip_opt() != 777 ) {
                for(int i =0; i < MAX_AS_ENTRIES; i ++) {
                        if(collector_[i].as_id == iph->ip_opt()){
                                collector_[i].num ++;
```

```
                                collector_[i].bytes += cmnh->size_;
                                break;
                        } else if(collector_[i].as_id == 0) {
                                collector_[i].as_id = iph->ip_opt();
                                collector_[i].num ++;
                                collector_[i].bytes += cmnh->size_;
                                break;
                        }
                }
        }
        return AddressClassifier::classify(p);
}
```

A.1.3 Link monitoring

In order to deal with the case that packets get dropped inside a domain, we need to monitor packet loss rate on a link. In ns, a link consists of multiple components, including enque and deque trace components. We add a counter in the enque and deque trace object to record how many packets (bytes) passed each of these two components. The loss rate on a link can be simply computed by dividing the difference of these two numbers by the number of bytes that enqueued.

```
void Trace::recv(Packet* p, Handler* h)
        ...
        format(type_, src_, dst_, p);

        //jungao
        hdr_cmn* cmnh = hdr_cmn::access(p);

        mib_.numPackets ++;
        mib_.numBytes += cmnh->size_;
        mib_.time = Scheduler::instance().clock();
        ...
}
```

And `mib_` has the type of `struct mib` which is defined in trace.h as follows:

```
struct mib {
        int numPackets;
        int numBytes;
        double time;
};
```

A QueryAgent runs on each node and retrieves packet loss information when receives request from Master BB agent. When a BB query queue message comes in, the QueryAgnet instance on a node will go through all the links the node(classifier) owns by following the agent's Classifier pointer and carefully go to the Enque and Deque trace object and retrieve the number of packets(bytes) that were enqueued and dequeued from their `mib_`. These information will then be sent back to the Master BB agent.

A.1.4 Bandwidth Broker

Bandwidth broker is implemented as an agent, `Class BBAgent`. BBAgent maintains an array of edge node BB agents and an array of QueryAgents for querying purpose.

1. **Initialization**

   The central data structure of a BBAgent is the matrix F, and it is defined as follows:

   ```
   // F matrix for the master BB
   struct row F[MAX_AS_ENTRIES];
   ```

   The related data structures are:

   ```
   //one hop(link) in the path
   struct path {
           struct path *next;

           // owner of the link
           int node_id_;

           // loss rate of this link
           double lossrate_;
   };

   // each data entry represents a flow from one ingress point to one
   // egress point
   struct data {
           // number of packets received
           int num;

           // number of bytes received
           int bytes;

           // the path for this ingress-egress pair
           struct path *hd;
           struct path *tl;
   };
   // one row of matrix F, each row represents one egress point
   struct row {

           // the egress point id
           int edge_id_;
           int node_id_;

           double measure_time_;
           struct data data_[MAX_AS_ENTRIES];
   };
   ```

BBAgent provides several commands for a simulation script to connect the master BB agent to agents on edge nodes and core nodes. The agents we run on edge nodes are also BBAgent, but they only need to retrieve and reply edge info to the Master BB. In the following command,

```
$b0 conn $b1 $b2 $b3 $b4
```

b0 is the master BB and b1-b4 are edge node BBs, and the command "conn" set up the connection between b0 and each of the edge BB. After the connections are set up, internally, master BB initializes all F matrix's entries to zero.

Similarly, BBAgent has another command "conn-query" to let the query agents on all nodes to have a chance to connect to the master BB. An example would be:

```
$b0 conn-query $q0 $q1
```

Here master BBAgent b0 connects to two query agents q0 and q1.

Command "retrieve-rt" allows master BB agent to retrieve paths from each ingress node to each egress node. The paths are then attached to each F entry. The following is used in one example script:

```
$ns at 0.0 "$b0 retrieve-rt"
```

Note that the "retrieve-rt" action should take place at the very beginning of a simulation so that F's entries get properly initialized.

2. **Communication**

A simple BB protocol was designed for exchanging messages between inter-domain BBs, between master BB and edge node BB, and between master BB and query agent. The protocol header is defined as follows:

```
struct hdr_bb {
        int type;
        int req_bw;
        int answer;
        nsaddr_t sender_addr;
        nsaddr_t receiver_addr;
        double send_time;
        int from_edge_id;
        int size;
        struct collector collector_[MAX_AS_ENTRIES];
};
```

The type of messages that a BB agent can send includes:

```
#define BB_MSG_REQUEST 0
#define BB_MSG_ANSWER  1
#define BB_MSG_QUERY 2
#define BB_MSG_REPLY 3

#define BB_MSG_QUERY_QUEUE 4
#define BB_MSG_REPLY_QUEUE 5
```

Currently, the master BB periodically sends out BB_MSG_QUERY messages to edge node BBs and sends BB_MSG_QUERY_QUEUE messages to query agents on core and edge nodes.

```
void BBAgent::sendit(int bw, int type) {

       // Create a new packet
       Packet* pkt = allocpkt();

       // Access the BB header for the new packet:
       hdr_bb* hdr = (hdr_bb*)pkt->access(off_bb_);

        hdr->type = type;

       hdr->req_bw = bw;

       hdr->sender_addr = this->addr_;
       hdr->receiver_addr = this->dst_;

       // Store the current time in the 'send_time' field
       hdr->send_time = Scheduler::instance().clock();

       // Send the packet
       send(pkt, 0);

       return;
}
```

BBAgent processes received messages in method `recv()`. The main structure of `recv()` is:

```
void BBAgent::recv(Packet* pkt, Handler*) {
        ...
        // Access the BB header for the received packet:
        hdr_bb* hdr = (hdr_bb*)pkt->access(off_bb_);
        ...
        switch (hdr->type) {

              case BB_MSG_QUERY:
                                   ...
              case BB_MSG_REPLY:
                                   ...
              case BB_MSG_REPLY_QUEUE:
                                   ...
              ...
              default:
        }
        ...
}
```

BBAgent updates matrix F when receives BB_MSG_REPLY and BB_MSG_REPLY_QUEUE messages. Message type BB_MSG_QUERY is for edge node BB agent to retrieve flow split information.

3. **Decision making**

A.2 Modifications to ns-2

- **node**

  One aspect of creating a topology in ns is creating nodes. The basic primitive of creating a node is:

```
set ns [new Simulator]
set n0 [$ns node]
```

  In order to create an edge node, we add a new argument to Node class's init method.

```
Node instproc init {is_edge args} {
        ...

        $self instvar edge_router_
        set edge_router_ $is_edge
        ...

        if {$edge_router_ == 0} {
                $self mk-default-classifier
        } else {
                $self mk-edge-classifier
        }

}
```

  To create an edge node, the following command is used in a tcl script:

```
set n0 [$ns node 1]
```

  Note that when the first argument is not 0, an edge node is created, and the non-zero argument represents the node's edge id. The difference between a regular node and an edge node is that the edge node's classifier object is an EdgeClassifier object as opposed to the default AddrClassifier object. `mk-edge-classifier` method is added to Class Node to attach an edge classifier to an edge node:

```
Node instproc mk-edge-classifier {} {

        $self instvar address_ classifier_ id_

        set classifier_ [new Classifier/Addr/Edge]
```

```
        # set up classifer as a router
        # (default value 8 bit of addr and 8 bit port)
        $classifier_ set mask_ [AddrParams set NodeMask_(1)]
        $classifier_ set shift_ [AddrParams set NodeShift_(1)]
        set address_ $id_

        $classifier_ set node_id_ $id_  ; # so we know who we are
        $classifier_ set edge_id_ [$self set edge_router_]
}
```

- **link**

  To complete a topology in an ns simulation, link objects are used to connect nodes together. For the purpose of marking packets with an edge id, we added a new Class SimpleEdgeLink, which is a subclass of SimpleLink. In addition to all the components that a SimpleLink has, a SimpleEdgeLink has one extra Stamper object inserted after the TTLChecker object. An edge link is one that connects an egress edge node of one domain with an ingress edge node of another domain. The edge id of the edge link should be the same as the edge id of the ingress edge node.

  ```
  Class SimpleEdgeLink -superclass SimpleLink

  SimpleEdgeLink instproc init { src dst bw delay edgeid q {lltype "DelayLink"} }

          eval $self next $src $dst $bw $delay $q $lltype ; # SimpleLink ctor

          $self instvar ttl_

          $self instvar stamper_  ; # not found in a SimpleLink

          set stamper_ [new Stamper]
          $stamper_ set edge_id_ $edgeid
          $stamper_ target [$ttl_ target]
          $ttl_ target $stamper_

  }
  ```

  A new method, `Simulator instproc simplex-edge-link { n1 n2 bw delay edgeid qtype args }` was added to the class Simulator to create edge links. This is how to create an edge link in a tcl script:

  ```
  $ns simplex-edge-link $n0 $n1 1Mb 10ms 101 DropTail
  ```

- **agent**

  Agents are placed on nodes to retrieve information and respond to BB agent's query. The Classifier object of a node contains traffic split information and packet loss information. In order for the agent to retrieve these information, when we create an agent and attach it to a node, we also set up a pointer in Agent object that points to this node's classifier object. This way, by using this pointer, an agent can retrieve the relevant information when needed.

```
Node instproc attach { agent { port "" } } {
        ...
#jungao
        #attach the current classifer of this node to the agent

#       puts "IN tcl node:=$self"

        $agent attach-classifier [$self set classifier_] [$self set
edge_router_

        #jungao
        #attach routing table to agent
        # $self (node)  knows the simulator that owns this simulation
        # retrieve routing table from $tmpns

        set tmpns [$self set ns_]
        $agent attach-rtb [$tmpns get-routelogic]
        ...
}
```

In this procedure, we also link the simulator's routing logic handle to the agent. The routing logic object contains the topology's routing table, this is used by the master BB agent to trace the path from any ingress point to any egress point.