TARIFF-BASED PRICING AND ADMISSION CONTROL FOR DIFFSERV NETWORKS

Tianshu Li, Youssef Iraqi and Raouf Boutaba
School of Computer Science
University of Waterloo, Canada
{dtianshu,iraqi,rboutaba}@bbcr.uwaterloo.ca

Abstract: In a QoS-Enabled network environment, there are two major concerns from both user's and provider's points of views: are there enough resources available for a particular traffic flow and what's the price for this flow? These two questions are exactly what admission control and pricing try to answer. An architecture that integrates pricing and admission control seems very promising. In this paper, we propose a tariff-based pricing architecture that integrates pricing and admission control for the DiffServ networks. We also study some pricing setting strategies for our architecture and evaluate our strategies through simulations.

Keywords: Pricing, Admission Control, DiffServ

1. Introduction

With the Internet evolving into a multi-service network, QoS-pricing in the Internet has been one of the hottest research areas in the recent years. Two QoS architectures: Integrated Services (IntServ)[1] and Differentiated Services (DiffServ)[2] have been standardized by the IETF to support QoS in the future Internet. Due to the inherent scalability problem of the IntServ approach, it is generally believed that DiffServ is more likely to be implemented in the Internet core. Unlike IntServ, which can charge users based on the allocated resources, pricing for DiffServ networks is more complicated and has drawn a lot of attention in the networking community. Before the wide deployment of DiffServ, an effective and efficient pricing scheme has to be developed. Meanwhile, since price is such an important economic incentive for the end users, pricing is often considered as an effective mechanism for congestion control and admission control, which in turn can improve the level of QoS guarantees. Indeed, QoS pricing schemes proposed so far often entail either congestion control or admission control or even both. In this paper, we first have a close look at the relationship between the pricing and these two traffic management functions and propose a tariff-based pricing architecture that integrates pricing and admission control for DiffServ networks. The proposed architecture maintains domain and global price tables for core networks only. In this way, we decouple the pricing for the core network from the end-to-end pricing, which fits well into the DiffServ paradigm.

The remainder of this paper is organized as follows: Section 2 will review some background and related work in this area. Section 3 discusses the motivation and some design choices and presents our pricing architecture and the construction and

maintenance of pricing tables. Section 4 and 5 discuss our price setting strategy and admission control scheme in details. Section 6 presents our simulation results and their analysis and finally section 7 concludes the paper.

2. Background and Related Work

Pricing for the Internet in general has long been an active research area. Example approaches such as proposed in [6–10] study the pricing for networks from various angles. More detailed review of pricing schemes can be found in [3, 4]. Most of approaches either assume a well-known user utility function or try to create a market environment for auctioning. In the first case, existence of a prior known utility function is assumed and price setting can be based on the optimization that maximizes either the social welfare globally or the user benefit locally. However, in [10], Shenker et al. argued that utility functions could not be well defined in short term and sometimes even very difficult in a long-term time scale. The effectiveness of such schemes is still questionable. Based on this observation, they proposed the Edge Pricing scheme that charges users for the estimated path and estimated cost so that pricing is pushed to the edge. In the second case, although auctioning does not require a prior knowledge of user traffic characteristics and has been generally considered as the one that achieves economic efficiency, it has significant implementation overhead. Up until now, there has not been a well-accepted solution yet.

Many of approaches mentioned above assume that users are rational to the price signals and have been using the pricing as a main mechanism for congestion control. When congestion occurs, extra congestion cost will be charged in order to address the externality issue. Since users are expected to react to the price signals, congestion-sensitive pricing schemes often emphasize user adaptation where users will adjust their sending rate in case of congestion. Some approaches that fall into this category can also be found in [12, 14].

However, in a strict sense, congestion-sensitive pricing does not address the QoS guarantee in particular. When congestion occurs, QoS is no longer guaranteed. To provide a better QoS guarantee, what we want is to avoid the congestion, not to act after the congestion occurs. Furthermore, QoS guarantee is a commitment that service providers made to the end users. Asking users to adapt to the price change or even terminate the service is undesirable. We believe that a proper interpretation for the user adaptation in a DiffServ environment is the choice of different service classes at the beginning of a service session rather than adjusting their sending rate in the middle of a service session. In other words, an elastic request would likely choose a lower level service class while an inelastic request may choose a higher level service class if the budget is sufficient. If the budget is not sufficient, then a request can either lower the service class requirement (if it is tolerable) or decide not to enter the network at all.

Another traffic management function that is closely related to the pricing is Admission control (AC). AC is often used to control the network load by restricting the access to the network and hence improve the level of QoS guarantee. Example approaches can be found in [13–16]. Studies also show that simple admission control algorithms based on estimated or measured network status are generally robust [15]. As being done in IntServ/RSVP architecture, admission control traditionally is performed at a hop-by-hop basis [1]. However, in a DiffServ environment, adding admission con-

trol functionality to all the core elements violate the DiffServ principle of keeping the core simple. End-point/edge admission control that pushes the admission control functionality into the edge of the network seems more suitable in this case. Most of the end-point AC approaches use probing [13, 16] or explicit congestion notification (ECN) [15, 16] to convey the network status back to the end points. A comprehensive study on endpoint admission control can be found in [16].

However, so far, most of the studies consider the pricing and admission control as two separate management functions. In other words, admission decisions are made solely on the load measurement or estimation and have no direct relation with the pricing. Using price as a primary admission criterion has not been studied sufficiently. In [15], authors suggest that admission decision could be made based on the user's willingness to pay for the ECN mark. However, it is not clear how users should pay for the mark (i.e. what is the price for the mark). In [12], Wang and Schulzrinne presented a complete pricing framework that integrates the admission control, congestion control, and pricing for DiffServ networks. However, they focus mainly on the congestion-sensitive pricing and do not study the admission control in details. Admission control in their framework is performed hop-by-hop and independently from pricing. Our architecture is similar to what they proposed but differs in a number of ways. This will be discussed in the subsequent sections in details.

3. Pricing Architecture

3.1 Motivation and Design Choices

From the discussion so far and the implication of our view of user adaptation, it is more desirable to tie the pricing with admission control in a DiffServ environment. Indeed, in a QoS-Enabled network environment, there are two major concerns from both user's and provider's points of views: are there enough resources available for a particular traffic flow and what's the price for this flow? These two questions are exactly what admission control and pricing try to answer. An architecture that integrates pricing and admission control is therefore very promising.

The design of our architecture is based on the following considerations.

- *Decouple the pricing for core networks from the access networks and end users:* Due to the scale and complexity of the Internet, a practical QoS implementation in the Internet is to deploy DiffServ in the core networks and give the access networks the freedom of implementing IntServ, DiffServ, or other QoS techniques [11]. This implies that a tightly integrated end-to-end pricing scheme is unlikely to be accepted. Furthermore, a large number of service providers are involved in the pricing of the Internet and each of them should have the choice of their own pricing scheme. A single pricing model that suits all is very impractical. Therefore, we emphasize a separate pricing scheme for the core networks that implements DiffServ technology. This decoupling enables us to focus on the network core only.
- *Price should reflect the availability of the network resources in the core:* As mentioned above, the ultimate goal of QoS is to avoid congestion. Therefore, it is more meaningful to charge users based on the remaining resources rather than the congestion cost. Since we are more interested in the implementation side of the problem, we do not assume a well known utility function for setting

the optimal price. In other words, each network element sets the price merely based on its own load. The rarer the resource, the higher the price. We also follow a simple and intuitive rule about price setting: price changes very slowly when there are plenty of available resource and increases drastically when the resource is scarce.

- *Per-flow messaging is not acceptable in a large network such as the Internet:* In order to aggregate and convey the price information to the access networks or even the end-users, we need a flexible and efficient architecture to accumulate the price along the path. In [12] a signaling protocol called resource negotiation and pricing protocol (RNAP) is proposed to accumulate the price along the path and negotiate the price and resource with the end-users. It is easy to see that the major drawback of such an approach is the per-flow messaging. Although the possibility of aggregating the messages has been investigated, the control message overhead is still significant. Our architecture tries to avoid this problem by maintaining the global price tables for the core networks at the access networks. This aggregates the pricing messages significantly.
- *Edge/end-point admission control fits well in a DiffServ environment:* Since the price reflects the availability of the resources in the core, admission decisions can mainly or even purely be based on the price (in this case, the price is the only admission criterion). Thanks to the decoupling of pricing for core networks, we are able to maintain the global price table at the access networks via domain abstraction. This enable us to push the admission control to the access networks. In our architecture, it is the end-users or access networks that decide whether a flow should enter the network or not. Inter-domain admission control is also possible using a global price table.

3.2 Price Table Construction

There are two types of price tables in our architecture: domain price table and global price table. A price entry in the domain price table represents the price for a service class from one edge node to another edge node within a domain, where a price entry in the global price table represents the price for a service class from one domain to another domain. Maintaining price information at such a scale may sound very impractical at a first glance. However, by abstracting an entire domain into a single node in a multi-domain network, constructing a global price table becomes feasible. Figure 1 depicts the concept of domain abstraction.

Semret et al. [9] also use a similar type of abstraction in their pricing scheme. A global market is created for all the domains to bid for the capacity in order to provide a QoS guaranteed service in their own domain. In their abstraction, overall capacity requirements in a domain is abstracted into a single bottleneck capacity. Although this is certainly a valid assumption, some inaccuracy is still introduced into the abstraction. In our abstraction, we do not make any simplification or assumption but rather accumulate the price for each actual ingress-egress path.

3.2.1 Domain price tables. Once each network element calculated the price locally, the price information has to be exchanged and aggregated over the entire domain. The ultimate goal here is to accumulate the price for each ingress-egress pair. Therefore, we need to know the route from the ingresses to the egresses. This requires

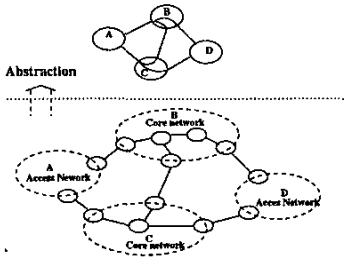

Figure 1. Domain Abstraction

Domain Price Table				
Route			Price	
Ingress	Egress	Intermediate		
a1	a5	a2, a4	class 1	45
			class 2	...
a1	a3	a2	...	
...	...	...	...	

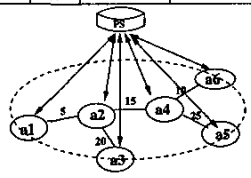

Figure 2. A Domain Price Table

some knowledge of the domain topology and routing table information within the domain. The choice of using a centralized or a decentralized approach largely depends on the routing strategy used in the domain.

If a link state routing approach such as OSPF or IS-IS is used, a centralized approach is preferred since all the information required to construct the domain price table is available immediately. Centralized approach uses a pricing station for each domain or autonomous system. The pricing station will communicate with all network elements within the domain and collect the price information. As a result, the pricing station will eventually maintain a price table for each ingress-egress pair. When an ingress-egress pair has multiple possible routes, the domain routing table is consulted and the route from the routing table will be used. Figure 2 depicts a sample domain price table maintained in a centralized pricing station (PS).

It is relatively complicated if the distance-vector routing approach such as RIP is used within the domain. A distributed approach can be considered in this case. One alternative is to add price as an extra set of metrics into the routing table, However, the major drawback of this approach is to bind the price update with the route update. A route update implies a price update but not the other way around. Furthermore, to propagate the price information among domains and construct the global price table, it's relatively easier when a centralized and complete view of the entire domain (e.g. a whole set of ingress-egress pairs) is available. One naive centralized approach in this case is to ask each element to send its routing table to the pricing station whenever the route update happens. Pricing station is then able to collect all the route information and generate the routes for each ingress-egress pair. A centralized pricing station is also well compatible with the Bandwidth Broker (BB)[5] approach. In fact, the pricing station can be part of the BB functionality. In the rest of the paper, we assume that a centralized pricing station is used. We also assume a pricing interval in our pricing architecture for both domain and global price tables. However, the update of the domain price table is not done periodically but is rather change driven to reduce the control-message overhead.

3.2.2 The Global price table. Because of the abstraction mentioned earlier, we are now able to construct the global price table without introducing too much over-

head.The price for an ingress-egress pair inside a domain becomes the price for passing through the node in the abstracted global network. Different routes that take different ingress-egress pairs through one domain will most likely have different prices. In this way, we can view the whole core networks as a single network that contains a limited number of nodes and links. Of course, further hierarchical decomposition can be applied if the size of global price table is still too large. In this paper, to give a simple and clear view of the idea, we assume only one level of abstraction and one level of global price table is constructed.

Unlike the domain price table, a global price table has to be constructed and maintained in a distributed manner. The update of the global price tables is also different. Periodical advertisement is used to propagate the price information as done in the Border Gateway Protocol (BGP). Each pricing station will advertise the price of a ingress-egress pair to its interested neighboring pricing station. Upon receiving such update information, each pricing station will update the global price table accordingly and propagate the update information to its interested neighbors at next pricing interval.

To deal with the issue of multiple routes through different domains, inter-domain routing tables are consulted during the price propagation. However, this time, we do not require a complete view of the route because the only information we want is the next domain or autonomous system which can be easily obtained from the BGP routing table (assuming that BGP is used for inter-domain routing). Figure 3 depicts the construction of global price table.

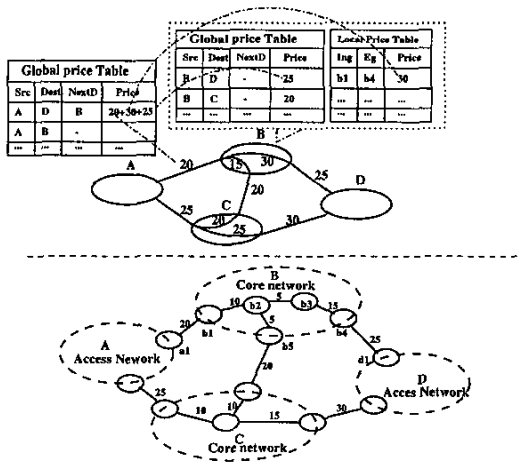

Figure 3. Global Price Table in Access networks

The possibility that paths in the routing table may change after the advertisement has a potential impact on the pricing. An admission decision based on the price of a route may become invalid if the path in the routing table changes (packets will eventually take a different route than the route they are expected to take and on which the admission decision was based). In this case, service quality may not be guaranteed especially if the new path does not have enough resources. Another concern is that

the price agreed by user and service provider is no longer valid. In this case, if there are enough resources in the new route, then we believe that service providers should absorb this price difference and simply continue the service. If the new route does not have enough resources, one possible solution is that service providers will notify the end-users to terminate the session without any charge. Since the price for the new path will be available "immediately", it is also possible to start a service renegotiation.

4. Price Setting Strategy

Now, the question left is how to set the price to reflect the availability of the network resources. Each network element will incorporate a load monitor so that price can be based on its current load level. Since network monitoring is out of the scope of this paper, we assume an existing method to monitor the traffic load for each service class.

We assume a basic unit price per unit time P_{unit} that can be computed offline for each service class. This basic unit price reflects the equipment costs, maintenance/administrative costs and business revenue consideration for a network element. The idea of differentiated service is to have a small number of classes where a higher price service class attracts less traffic and consequently have a better QoS guarantee. Hence, we consider a simple and practical approach where service providers set up a targeted capacity fill factor f_i for each class to enable the service differentiation between the classes(i.e. $f_i = T_i/C^i_{max}$, where T_i is the targeted capacity for class i and C^i_{max} is the maximum capacity for class i). Therefore, it is straightforward to see that the base price for a service class P^i_{base} is inversely proportional to this factor.

$$P^i_{base} = P_{unit}/f_i \tag{1}$$

To compute the dynamic price for a service class, Wang and Schulzrinne adopt an iterative tatonnement process in [12]:

$$P_i(t) = P_i(t-1) + a_i * (D_i - T_i)/T_i$$

Where $P_i(t)$ denotes the price for class i at time t, and D_i is the demand or current load for class i and a_i is the convergence rate factor. However in their case, the price changes gradually and can not successfully reflect the real traffic condition inside the network. We believe that when the network is severely stressed the price should increase much faster. As mentioned above, we follow an intuitive way for the price setting. Figure 4 illustrates our pricing strategy in general. When the load for a particular service class is lower than its targeted capacity, the price is simply the base price P^i_{base} for a particular service class. When the load exceeds its target capacity the price will be increased rapidly and even dramatically when the load is close to the maximum capacity. we adopt the exponential growth in this case to reflect our strategy of price setting:

$$P_i(t) = \begin{cases} P^i_{base} & \text{if } D_i(t) \leq T_i \\ P^i_{base}\, e^{\alpha_i[\frac{D_i(t)}{T_i}-1]} & \text{otherwise} \end{cases} \tag{2}$$

Note that when demand exceeds C_{max}, we can simply set the price to ∞ to indicate that there is no longer available resource for new requests. Admission control can be enforced in this situation. Alternatively, there can be no price limit even if the demand exceeds the maximum capacity. P^i_{max} then is not the price upper bound but rather the

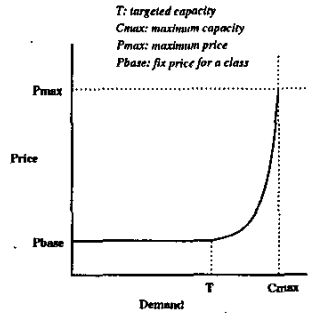

Figure 4. General price setting strategy

price when the demand reaches the maximum capacity of that particular service class. Since the price increases exponentially and becomes so high that we expect no user would actually accept the price.

There are several ways to decide the α_i factor. Here, we consider a simple case where P^i_{max} is known. One can use the base price for class $i+1$ as the P^i_{max} for class i, or alternatively decide the P^i_{max} based on some business considerations, which is more likely the case in the real world. In the later case, P^i_{max} is allowed to be greater than the P^{i+1}_{base}, which enables the switching between service classes when the price for a lower level service class is higher than the price for upper level service class and hence balances the load among service classes.

Now, assuming that a maximum price P^i_{max} is available for each service class, Since $f_i = T_i/C^i_{max}$, we can obtain the α_i factor by solving the following equation.

$$P^i_{max} = P^i_{base}\, e^{\alpha_i[\frac{C^i_{max}}{T_i}-1]} \Longrightarrow \alpha_i = \log(\frac{P^i_{max}}{P^i_{base}}) * (\frac{f_i}{1-f_i}) \qquad (3)$$

5. End-to-End Pricing and Admission Control

5.1 End-to-End Pricing

One of the main advantages of our pricing framework is that it enables us to focus on the pricing in the core networks only. Depending on the particular technology implemented in the access networks, various pricing schemes can be applied to achieve the end-to-end pricing. For example, access networks can implement flat rate pricing or time of day pricing for their simplicity and predictability. In this case, the cost of accessing core networks could be absorbed by the access networks and it is the access networks that choose an appropriate service class for the user traffic based on some policy defined by the access networks. Alternatively, access networks can charge user for the reserved resource if IntServ is implemented or even use the same pricing scheme as in the core if DiffServ is used.

Since the pricing is strictly for the network core and global price tables are available at the access networks, end-to-end pricing can be implemented without involving the core nodes at all. This eliminates possible scalability problems caused by pricing

and admission control signaling. it also gives the access networks the flexibility of implementing an end-to-end pricing scheme in different protocol layers. For example, if users or access networks are really keen on the exact end-to-end pricing, a light weight signaling protocol can be used between the sender and receiver access networks without any involvement of the core nodes. This protocol can be implemented in the network layer as the pricing for core networks does or even in the upper layers such as the transport layer or the application layer.

5.2 Admission Control

Another desirable property of our pricing framework is that the admission control decision can be made based on this price information since it effectively reflects the availability of the path. Our admission control algorithm then consists of the following two parts when a flow request is generated:

1 Lookup the price for a service class in the global price table. End-users or access networks decide whether the price is acceptable or not based on their budget constraints and other service level agreement (SLA) parameters. If the price can not be accepted then check the possibility of switching among service classes (with possibly different service requirements).

2 If the advertised price is accepted, then final admission control decision is made by the service providers based on the estimated bottleneck traffic load l_{max} against the targeted threshold for the service class. This deals with the situation that when users are willing to pay for the service but there is no resource available.

Ideally, we want to know the relationship between the total price and the available resource at the bottleneck link. In other words, we are most interested in the relationship between the total price $P^i_{total}(t)$ and the load of bottleneck link. From section 4 we know that the accumulated price at time t for class i observed at the edge is:

$$P^i_{total}(t) = \sum P^{ij}_{base} + \sum P^{ik}_{base}\, e^{\alpha_{ik}\,[\frac{D_{ik}(t)}{T_{ik}}-1]}, (D_{ij}(t) \leq T_{ij}, D_{ik}(t) > T_{ik})$$
$$\text{where } D_{ij}(t) \text{ is the demand for class } i \text{ at link } j \text{ at time } t \quad (4)$$

Given only the value of $P^i_{total}(t)$, it is almost impossible to obtain the exact value of the $\max[D_{ij}(t)/T_{ij}]$. However, in our pricing scheme, $P_i(t)$ at the bottleneck link becomes the dominant term in the $P^i_{total}(t)$ when it reaches a fairly high level. This is mainly because of the exponential growth of $P_i(t)$ and it enables us to estimate the value of the $\max[D_{ij}(t)/T_{ij}]$ with much less error.

For the sake of understanding, in the rest of this section, all notations are for service class i unless otherwise specified. Let $l_j(t)$ denotes the traffic load at link j at time t (i.e. $l_j(t) = D_j(t)/T_i$), and $l_{max}(t)$ denotes the load for the link that has the highest load along the entire path). From equation 4, we have

$$P_{total}(t) = \sum P^j_{base} + \sum P^k_{base}\, e^{\alpha_k\,(l_k(t)-1)} + P^h_{base}\, e^{\alpha_h\,(l_{max}(t)-1)} \quad (5)$$

where h is the link that has the highest load $l_{max}(t)$ at time t along the path and the last term in the equation is the price for the bottleneck link h. The fist term is the sum of the price for all the links that have the base prices and the second term is the sum of the price for the rest of the links respectively. Then a bound is given by,

$$P_{total}(t) \geq \sum_{all} P_{base} - P^h_{base} + P^h_{base}\, e^{\alpha_h (l_{max}(t)-1)} \tag{6}$$

where $\sum_{all} P_{base}$ is the sum of the P_{base} for all the links along the path, which is denoted as P^{total}_{base}. By rearranging the terms in the equation and plugging in the value of α_h from equation 3, we can see that $l_{max}(t)$ must satisfy the following relation.

$$l_{max}(t) \leq 1 + \frac{\log\left(\frac{P_{total}(t) - P^{total}_{base}}{P^h_{base}} + 1\right)}{\log\left(P^h_{max}/P^h_{base}\right) * \left(\frac{f_h}{1-f_h}\right)} \tag{7}$$

We will use the $\min\{f_h\}$, $\max\{P^h_{base}\}$, and $\min\{P^h_{max}\}$ respectively and use f, P_{base} and P_{max} to denote them. Since they are constant for the path, they can be collected and computed offline.

For the purpose of admission control, using upper bound often provides quite conservative results. To examine the relationship between $P_{total}(t)$ and $l_{max}(t)$, we first consider the case where there is a single bottleneck link along the path. Clearly, the upper bound value obtained above will be very close to the exact $l_{max}(t)$ because the rest of the terms in equation 5 are indeed ignorable. However, it is more complicated when there are multiple bottleneck links along the path. Some estimations are required to handle this situation. Since we expect that f_h, P^h_{max}, and P^h_{min} vary within a relatively small range, without loss of generality, we can rewrite equation 6 into

$$P_{total}(t) \simeq P^{total}_{base} - r * P_{base} + r * P_{base}\, e^{\alpha\,(l_{max}(t)-1)} \tag{8}$$

where r is the number of the bottleneck links along the path that all should be taken into consideration. Note that we are using the f, P_{base}, and P_{max}. However, in this case, they are not necessarily the maximum or minimum values since a number of links are involved. Further refining such as average or weighted average can be applied depending on the specific network condition. Solving the above equation gives us

$$l_{max}(t) \simeq 1 + \frac{\log\left(\frac{P_{total}(t) - P^{total}_{base}}{r * P_{base}} + 1\right)}{\log\left(P_{max}/P_{base}\right) * \left(\frac{f}{1-f}\right)} \tag{9}$$

It is then clear that l_{max} most likely lies between the upper bound and the above estimated value. It is difficult to estimate the value of r without any knowledge of network topology or explicit messaging. However, for a long time scale, it is possible that each domain is able to identify the approximate number of bottleneck links and the probability of multiple bottlenecks existing inside the domain. In this way, one can compute a weighted average $\overline{r}$ value for each ingress-egress pair and exchange them offline. Of course, one can always set r to 1, which is the case of upper bound admission control. In this case, the difference of these two values also indicates the error range of our estimation. We should also notice that the error caused by the incorrect value r is reduced significantly because of the logarithmic nature of our estimation.

6. Simulation and Results

6.1 Simulation Model

In order to study the behavior of our pricing strategy, we setup a DiffServ network environment using the *ns 2* simulator. We modified the DiffServ implementation in *ns* developed by Nortel Networks to incorporate our pricing and admission control mechanisms. Since the goal of the simulation is to evaluate our price setting strategy and admission control scheme, we only simulate a single DiffServ domain and do not focus on the construction and maintenance of the domain and global price tables.

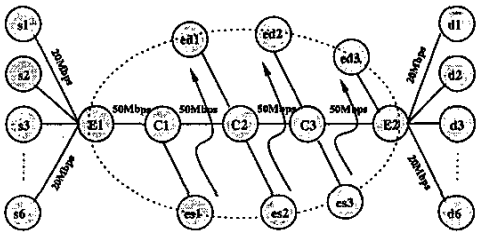

Figure 5. Simulation Network Topology

Figure 5 illustrates the network topology used in our simulation which consists of three core routers and eight edge routers. Three core routers implement the dsRED core queue which has no policing and marking functionality but only PHB forwarding. All edge routers implement the dsRED edge queue which supports the DiffServ packet classifying, marking, and policing. Edge router E1 acts as the pricing station and handles user requests generated at sources. Additionally, six extra edge routers are configured inside the DiffServ domain to create the cross traffic and simulate bottlenecks at link C1C2, C2C3, and C3E2.

The total capacity of each link from source nodes to the edge of DiffServ domain is set to 20Mbps, and 50 Mbps for all links within the DiffServ domain. Propagation delay for all the links are set to 5ms. All links are full duplex outside the DiffServ domain and DropTail queue management is used in this case. Inside the DiffServ domain, only the links that connect core routers are full duplex and the rest of the links are simplex links because different type of dsRED queuing techniques are used in different directions. Weighted Round Robin (WRR) scheduling is used in each link. In our simulation we consider three service classes and the weights for the three classes are distributed as 3, 3, and 4, and the expected load for each class is set to 50%, 70%, and 90% respectively. The base price P^i_{base} and full load price P^i_{max} for each class are set as $0.16/0.7(\$)$, $0.09/0.35$ (\$), and $0.04/0.16(\$)$ per time unit respectively. A pricing agent is attached to each link to set the price locally and communicate with each other to propagate the price information back to the edge. The admission threshold is set to 0.75, 0.9, and 1.1 respectively.

For each class, there are two types of traffic sources in our simulation. CBR and Pareto on/off traffic are generated independently to the edge E1 and flow requests are modelled by a Poisson arrival distribution. The holding time for each flow is exponentially distributed with a mean value of $150s$. The average rate for CBR is

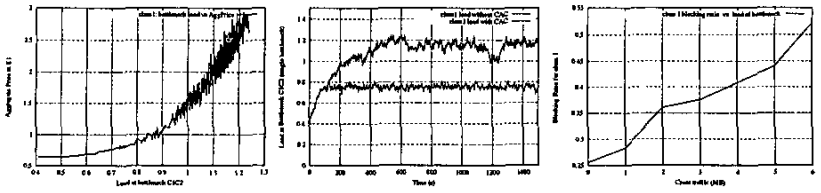

Figure 6. (single bottleneck a→c from left to right) (a)aggregated Price at E1 vs. Load at C1C2 (b)load at C1C2 without/with CAC (c)request blocking ratio vs. Traffic(Mb) at C1C2

128k. For the Pareto traffics, the shape parameter is set to 1.5, where the on and off time are both set to 500ms, and the peak rate is set to 128K. The rest of the parameters are set to the default values in *ns*. The total time for each run is 1500s.

6.2 Result Analysis

6.2.1 Single Bottleneck. We first consider the case of single bottleneck to examine the basics of our approach. The rate of cross traffic at C1C2 is set to be approximately 40% of the class 1 capacity at C1C2. The rate of the other two cross traffics are set to be less than 10% of the class 1 capacity accordingly at C2C3 and C3E3. Therefore, C1C2 will be the only bottleneck along the path.

Figure 6a gives the aggregate price observed at E1 vs. the load at C1C2 when admission control is not used. As we expected, it is a well shaped exponential curve because there is only one bottleneck and its price dominates the total price observed at E1. To test the effectiveness of our admission control algorithm, we first run the experiment without admission control and then repeat the experiment with admission control under the same traffic condition. Figure 6b shows that our estimation of network load based on the price is indeed accurate in the case of single bottleneck. The load at the bottleneck link is well controlled at about 0.75, which is the admission threshold preselected for class 1.

We repeat the simulation with different sending rates of cross traffic at C1C2. As expected, the load at C1C2 are all well controlled but with different request blocking ratios. Figure 6c shows the request blocking ratio for all the sources vs. the sending rate of cross traffic at C1C2.

Throughout the experiment, we experience very few packet losses except when the total sending rate of flows for all service classes exceeds the total capacity of the bottleneck. This is mainly due to the use of WRR scheduling and no individual dropping enforced for each class. When a class load exceeds its class capacity, it tends to steal the bandwidth from other service classes. Since we are mostly concerned about controlling the traffic load and keeping it lower than a threshold, we do not present the packet loss and delay results here. Ideally, by carefully setting the admission control threshold for each service class, we should not see any packet loss.

6.2.2 Multiple Bottlenecks. To simulate a multiple bottlenecks situation, we increase the sending rates of cross traffics at C2C3 and C3E2 to about 35% so that they are close to but still lower than the sending rate of cross traffic at C1C2. There-

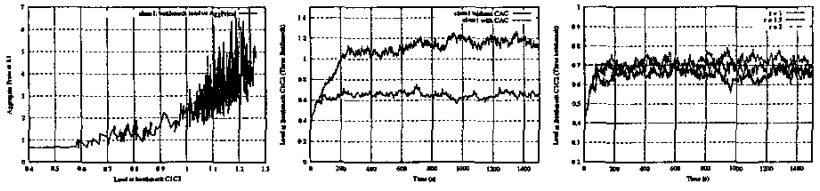

Figure 7. (three bottlenecks, a→c from left to right)(a) aggregated Price at E1 vs load at C1C2 (b) Load at bottleneck C1C2 without/with CAC (c) Load at bottleneck C1C2 with different r values

fore, three bottlenecks are simulated in our network. As in the singe bottleneck case, we first run the simulation without admission control and then repeat the simulation with admission control.

Figure 7a shows the aggregated price observed at E1 vs. the load at bottleneck C1C2 when admission control is not enforced. Because no single bottleneck can dominate the price for the entire path, the aggregated price is a little harder to predict when the links are heavily loaded. Fortunately, the fluctuation of the price does not affect the effectiveness or stability of our admission control algorithm. Figure 7b shows that the traffic load at bottleneck link C1C2 is also well controlled consistently. We do observe that the estimated load is not as accurate as in the single bottleneck. This is mainly because we are using the upper bound admission control and the result is expected to be conservative. The simulation result shows a fairly close traffic load estimation (0.7 vs. 0.75). Throughout the experiment, we can see that our approach has a very consistent performance. This indicates that our admission control approach is robust and the fluctuation of price would not affect the stability of our approach.

Based on the discussion in section 5.2, we also vary the factor r to see how it will affect our simulation results. Figure 7c shows three sets of load at bottleneck C1C2 throughout the experiment with r set to 1, 1.5, and 2. As we can see, r has a small but positive impact on the load estimation. In other words, choosing a close enough r value could indeed improve the efficiency of our admission control approach. However, the improvement is quite small. This is because the error of our estimation is reduced significantly by the logarithmic nature of our load estimation algorithm.

7. Conclusion

The main objective of having the global price table at the level of access networks is to enable an accurate and fast decision-making process. In this paper, we presented our approach to the pricing in DiffServ networks and proposed a pricing architecture that separates the pricing for core networks from the end-to-end pricing through domain abstraction and maintaining domain and global price tables. We also described our pricing strategy and suggested an associated admission control mechanism. Since the price in our scheme effectively reflects the resource availability inside the network, it is used not only as an economic incentive but also as a mean of estimating the bottleneck traffic load.

Our architecture is flexible in the sense that end-to-end pricing is decoupled from the network core and scalable thanks to the domain abstraction. Admission control is

pushed to the edge and no per-flow based messaging for either pricing or admission control is needed. This way, our architecture follows the philosophy of the edge-pricing scheme but with better network utilization and QoS guarantee. Maintaining the price tables for core networks also enables the split of revenue among the service providers.

We believe that the benefit gained from maintaining the price tables can certainly overweight the overhead it introduces. Our future work includes developing a user-behavior model that models the user reaction to price change and studying the impact it may have on the performance of our admission control mechanism. We are also investigating the applicability of our architecture in other contexts. For example, QoS routing using price as a constraint may be an interesting application of our pricing architecture since the price is available immediately and reflects the availability of resources inside the network.

References

[1] S. Shenker, et al, *"Specification of guaranteed quality of service,"* RFC 2212, IETF, Sept. 1997.

[2] S. Blake et al, *"An Architecture for Differentiated Services,"* RFC 2475, IETF, Dec. 1998.

[3] M. Falkner, M. Devetsikiotis, and I. Lambadaris, *"An Overview of Pricing Concepts for Broadband IP Networks,"* IEEE Communications Surveys & tutorials, 2nd Quarter, pp.9-13, 2000.

[4] L.A.DaSilva, *"Pricing for QoS-enabled Networks: A Survey,"* IEEE Communications Surveys & Tutorials, 2nd Quarter, pp.2-8, 2000.

[5] B. Teitelbaum et al. *"Internet2 QBone: building a testbed for differentiated services,"* IEEE Network, vol.13, no.5, pp.8-17, September 1999.

[6] A. M. Odlyzko, *"Paris Metro Pricing for the Internet,"* Proceedings of the ACM Conference on Electronic Commerce, pp.140-147, 1999.

[7] J. MacKie-Mason, L. Murphy, and J. Murphy, *"Responsive Pricing in the Internet,"* Internet Economics, L. W. McKnight and J.P. Bailey, Eds., Cambridge, Massachusetts, MIT Press, pp.279-303, 1997.

[8] J. K. MacKie-Mason, and H. R. Varian, *"Pricing Congestible Network Resources,"* Selected Areas in Communications, IEEE Journal on, vol.13, no.7, pp.1141-1149, 1995.

[9] N. Semret, R.R.-F. Liao, A.T. Campbell, and A.A. Lazar, *"Pricing, provisioning and peering: dynamic markets for differentiated Internet services and implications for network interconnections,"* Selected Areas in Communications, IEEE Journal on, vol.18, no.12, pp.2499 -2513, Dec. 2000 .

[10] S. Shenker, D. Clark, D. Estrin, and S. Herzog, *"Pricing in Computer Networks: Reshaping the Research Agenda,"* ACM Computer Communication Review, vol. 26, no. 2, pp. 19-43, April 1996.

[11] V. Fineberg, *"A Practical Architecture for Implementing End-to-End QoS in an IP Network,"* IEEE Communications Magazine, vol. 40, no.1, pp.122 -130, Jan. 2002.

[12] X. Wang and H. Schulzrinne, *"Pricing Network Resources for Adaptive Applications in a Differentiated Services Network,"* In Proceeding of INFOCOM 2001, Anchorage, Alaska, April 2001.

[13] Bin Pang, Huairong Shao, Wenwu Zhu, and Wen Gao, *"An admission control scheme to provide end-to-end statistical QoS provision in IP networks,"* Performance, Computing, and Communications Conference, 21st IEEE International, pp.399 -403, 2002.

[14] A. Ganesh and K. Laevens, *"Congestion Pricing and User Adaptation,"* in Proceedings IEEE INFOCOM, Anchorage, USA, April 2001.

[15] F. P. Kelly, P. B. Key, and S. Zachary, *"Distributed admission control, "* IEEE Journal on Selected Areas in Communications, Special Issue on Internet QoS, pp.2617Ú2628, Dec. 2000.

[16] L. Breslau, E. W. Knightly, S. Schenker, I. Stoica, and H. Zhang, *"Endpoint Admission Control: Architectural Issues and Performance,"* ACM SIGCOMM 2000, Stockholm, Sweden, August 2000.