Distributed Dynamic Capacity Contracting: A congestion pricing framework for Diff-Serv *

Murat Yuksel[1] and Shivkumar Kalyanaraman[2]

[1] CS Department, Rensselaer Polytechnic Institute,
110 8th Street, Troy, NY 12180, USA
yuksem@cs.rpi.edu

[2] ECSE Department, Rensselaer Polytechnic Institute,
110 8th Street, Troy, NY 12180, USA
shivkuma@ecse.rpi.edu

Abstract. In order to provide better Quality-of-Service (QoS) in large networks, several congestion pricing proposals have been made in the last decade. Usually, however, those proposals studied optimal strategies and did not focus on implementation issues. Our main contribution in this paper is to address implementation issues for congestion-sensitive pricing over a single domain of the differentiated-services (diff-serv) architecture of the Internet. We propose a new congestion-sensitive pricing framework Distributed Dynamic Capacity Contracting (Distributed-DCC), which is able to provide a range of fairness (e.g. max-min, proportional) in rate allocation by using pricing as a tool. Within the Distributed-DCC framework, we develop an Edge-to-Edge Pricing Scheme (EEP) and present simulation experiments of it.

1 Introduction

As multimedia applications with extensive traffic loads are becoming more common, better ways of managing network resources are necessary in order to provide sufficient QoS for those multimedia applications. Among several methods to improve QoS in multimedia networks and services, one particular method is to employ congestion pricing. The main idea is to increase service price when network congestion is more, and to decrease the price when congestion is less.

Implementation of congestion pricing still remains a challenge, although several proposals have been made, e.g. [1, 2]. Among many others, two major implementation obstacles can be defined: need for *timely feedback* to users about price, determination of *congestion information* in an efficient, low-overhead manner.

The first problem, timely feedback, is relatively very hard to achieve in a large network such as the Internet. In [3], the authors showed that users do need feedback about charging of the network service (such as current price and prediction of service quality in near future). However, in our recent work [4], we illustrated that congestion control through pricing cannot be achieved if price changes are performed at a time-scale larger than roughly 40 round-trip-times (RTTs), which is not possible to implement for many cases. We believe that the

* This work is sponsored by NSF under contract number ANI9819112, and co-sponsored by Intel Corporation.

problem of timely feedback can be solved by placing intelligent intermediaries (i.e. software or hardware agents) between users and service providers. In this paper, we do not focus on this particular issue and leave development of such intelligent agents for future research.

The second problem, congestion information, is also very hard to do in a way that does not need a major upgrade at network routers. However, in diff-serv [5], it is possible to determine congestion information via a good ingress-egress coordination. So, this flexible environment of diff-serv motivated us to develop a pricing scheme on it.

In our previous work [6], we presented a simple congestion-sensitive pricing "framework", *Dynamic Capacity Contracting (DCC)*, for a single diff-serv domain. DCC treats each edge router as a station of a service provider or a station of coordinating set of service providers. Users (i.e. individuals or other service providers) make *short-term contracts* with the stations for network service. During the contracts, the station receives congestion information about the network core at a time-scale smaller than contracts. The station, then, uses that congestion information to update the service price at the beginning of each contract. Several pricing "schemes" can be implemented in that framework.

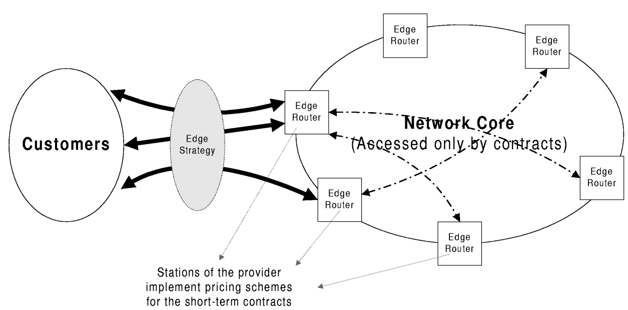


Fig. 1. DCC framework on diff-serv architecture.

DCC models a short-term contract for a given traffic class as a function of price per unit traffic volume P_v, maximum volume V_{max} (maximum number of bytes that can be sent during the contract) and the contract length T:

$$Contract = f(P_v, V_{max}, T) \tag{1}$$

Figure 1 illustrates the big picture of DCC framework. Customers can only access network core by making contracts with the provider stations placed at the edge routers. The stations offer contracts (i.e. V_{max} and T) to fellow users. Access to these available contracts can be done in different ways, what we call *edge strategy*. Two basic edge strategies are "bidding" (many users bids for an available contract) or "contracting" (users negotiate P_v with the provider for an available contract). So, edge strategy is the decision-making mechanism to identify which customer gets an available contract at the provider station.

However, in DCC, we assumed that all the provider stations advertise the same price value for the contracts, which is very costly to implement over a wide area network. This is simply because the price value cannot be communicated to all stations at the beginning of each contract. In this paper, we relax this assumption by letting the stations to calculate the prices locally and advertise different prices than the other stations. We call this new version of DCC as *Distributed-DCC*. We introduce ways of managing the overall coordination of the stations for the common purposes of fairness and stability. We then develop a pricing scheme Edge-to-Edge Pricing (EEP). We illustrate stability of EEP by simulation experiments. We address fairness problems related to pricing, and show that the framework can achieve max-min and proportional fairness by tuning a parameter, called as *fairness coefficient*.

The paper is organized as follows: In the next section, we position our work and briefly survey relevant work in the area. In Section 3 we describe Distributed-DCC framework, and investigate various issues, such as price calculation, fairness, scalability. Next in Section 4, we develop the pricing scheme EEP. In Section 5, we make experimental comparative evaluation of EEP. We finalize with summary and discussions.

2 Related Work

There has been several pricing proposals, which can be classified in many ways such as *static* vs. *dynamic*, *per-packet* charging vs. *per-contract* charging.

Although there are opponents to dynamic pricing in the area (e.g. [7, 8]), most of the proposals have been for dynamic pricing (specifically congestion pricing) of networks. Examples of dynamic pricing proposals are MacKie-Mason and Varian's Smart Market [1], Gupta et al.'s Priority Pricing [9], Kelly et al.'s Proportional Fair Pricing (PFP) [10], Semret et al.'s Market Pricing [11], and Wang and Schulzrinne's Resource Negotiation and Pricing (RNAP) [12, 2]. Odlyzko's Paris Metro Pricing (PMP) [13] is an example of static pricing proposal. Clark's Expected Capacity [14] and Cocchi et al.'s Edge Pricing [15] allow both static and dynamic pricing. In terms of charging granularity, Smart Market, Priority Pricing, PFP and Edge Pricing employ per-packet charging, whilst RNAP and Expected Capacity do not employ per-packet charging.

Smart Market is based primarily on imposing per-packet congestion prices. Since Smart Market performs pricing on per-packet basis, it operates on the finest possible pricing granularity. This makes Smart Market capable of making ideal congestion pricing. However, Smart Market is not deployable because of its per-packet granularity and its many requirements from routers. In [16], we studied Smart Market and difficulties of its implementation in more detail. While Smart Market holds one extreme in terms of granularity, Expected Capacity holds the other extreme. Expected Capacity proposes to use *long-term* contracts, which can give more clear performance expectation, for statistical capacity allocation and pricing. Prices are updated at the beginning of each long-term contract, which incorporates little dynamism to prices. Our work, Distributed-DCC, is a middle-ground between Smart Market and Expected Capacity in terms of granularity. Distributed-DCC performs congestion pricing at *short-term* contracts, which allows more dynamism in prices while keeping pricing overhead small.

Another close work to ours is RNAP, which also mainly focused on implementation issues of congestion pricing on diff-serv. Although RNAP provides a complete picture for incorporation of admission control and congestion pricing, it has excessive implementation overhead since it requires all network routers to participate in determination of congestion prices. This requires upgrades to all routers similar to the case of Smart Market. Our work solves this problem by requiring upgrades only at edge routers rather than at all routers.

3 Distributed-DCC: The Framework

Distributed-DCC is specifically designed for diff-serv architecture, because the edge routers can perform complex operations which is essential to several requirements for implementation of congestion pricing. Each edge router is treated as a station of the provider. Each station advertises locally computed prices with information received from other stations. The main framework basically describes how to preserve coordination among the stations such that stability and fairness of the overall network is preserved. A *Logical Pricing Server* (LPS) (which can be implemented in a centralized or distributed manner, see Section 3.6) plays a crucial role. Figure 2 illustrates basic functions (which will be better understood in the following sub-sections) of LPS. The following sub-sections investigate several issues regarding the framework.

3.1 How to Calculate p_{ij} ?

Each ingress station i keeps a "current" price vector p_i, where p_{ij} is the price for flow from ingress i to egress j. So, how do we calculate the price-per-flow, p_{ij}? The ingresses make estimation of budget for each edge-to-edge flow passing through themselves. Let $\hat{b}_{ij}$ be the currently *estimated budget* for flow f_{ij} (i.e the flow from ingress i to egress j). The ingresses send estimated budgets to the corresponding egresses (i.e. $\hat{b}_{ij}$ is sent from ingress i to egress j) at a deterministic time-scale. At the other side, the egresses receive budget estimations from all the ingresses, and also they make estimation of capacity $\hat{c}_{ij}$ for each particular flow. In other words, egress j calculates $\hat{c}_{ij}$ and is informed about $\hat{b}_{ij}$ by ingress i. Egress j, then, penalizes or favors f_{ij} by updating its estimated budget value, i.e. $b_{ij} = f(\hat{b}_{ij}, [parameters])$ where $[parameters]$ are optional parameters used for deciding whether to penalize or favor the flow. For example, if f_{ij} is passing through more congested areas than the other flows, egress j can penalize f_{ij} by reducing its budget estimation $\hat{b}_{ij}$.

At *another time-scale*[3], egresses keep sending information to LPS. More specifically, for a diff-serv domain with n edge routers, egress j sends the following information to LPS:

[3] Can be larger than ingress-egress time-scale, but should be less than contract length.

1. the *updated budget estimations* of all flows passing through itself, i.e. b_{ij} for $i = 1..n$ and $i \neq j$
2. the *estimated capacities* of all flows passing through itself, i.e. $\hat{c}_{ij}$ for $i = 1..n$ and $i \neq j$

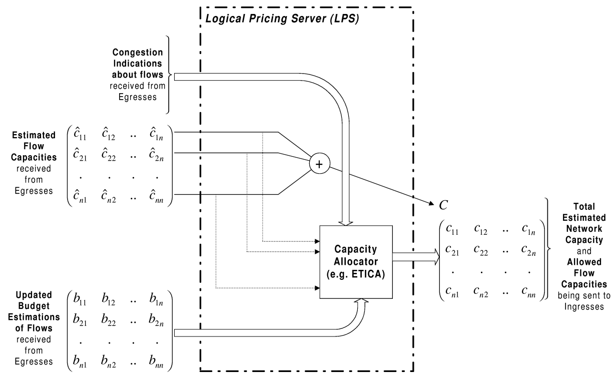


Fig. 2. Major functions of LPS.

LPS receives information from egresses and, for each f_{ij}, calculates *allowed capacity* c_{ij}. Calculation of c_{ij}s is a complicated task which depends on b_{ij}s. In general, the flows should share capacity of the same bottleneck in proportion to their budgets. We will later define a generic algorithm to do capacity allocation task. LPS, then, sends the following information to ingress i:

1. the *total estimated network capacity* C (i.e. $C = \sum_i \sum_j \hat{c}_{ij}$)
2. the allowed capacities to each edge-to-edge flow starting from ingress i, i.e. c_{ij} for $j = 1..n$ and $j \neq i$

Now, the pricing scheme at ingress i can calculate price for each flow by using c_{ij} and $\hat{b}_{ij}$. An example pricing scheme will be described in Section 4.

3.2 Budget Estimation at Ingresses

In order to determine user's real budget The ingress stations perform very trivial operation to estimate budgets of each flow, $\hat{b}_{ij}$. The ingress i basically knows its current price for each flow, p_{ij}. When it receives a packet it just needs to determine which egress station the packet is going to. Given that the ingress station has the addresses of all egress stations of the same diff-serv domain, it can find out which egress the packet is going to. So, by monitoring the packets transmitted for each flow, the ingress can estimate the budget of each flow. Let x_{ij} be the total number of packets transmitted for f_{ij} in unit time, then the budget estimate for f_{ij} is $\hat{b}_{ij} = x_{ij} p_{ij}$. Notice that this operation must be done at the ingress rather than egress, because some of the packets might be dropped before arriving at the egress. This causes x_{ij} to be measured less, and hence causes $\hat{b}_{ij}$ to be less than it is supposed to be.

3.3 Congestion-Based Capacity Estimation at Egresses

The essence of capacity estimation in Distributed-DCC is to decrease the capacity estimation when there is congestion indication(s) and to increase it when there is no congestion indication. This will make the prices *congestion-sensitive*, since the pricing scheme is going to adjust the price according to available capacity. In this sense, several capacity estimation algorithms can be used, e.g. Additive Increase Additive Decrease (AIAD), Additive Increase Multiplicative Decrease (AIMD). We now provide a full description of such an algorithm.

With a simple congestion detection mechanism (such as marking of packets at interior routers when congested), egress stations make a congestion-based estimation of the capacity for the flows passing through themselves. Egress stations divide time into deterministic *observation intervals* and identify each observation interval as *congested* or *non-congested*. Basically, an observation interval is congested if a congestion indication was received during that observation interval. At the end of each observation interval, the egresses update the estimated capacity. Then, egress j calculates the estimated capacity for f_{ij} at the end of observation interval t as follows:

$$\hat{c}_{ij}(t) = \begin{cases} \beta * \mu_{ij}(t), \textit{ congested} \\ \hat{c}_{ij}(t-1) + \triangle\hat{c}, \textit{ non-congested} \end{cases}$$

where β is in (0,1), $\mu_{ij}(t)$ is the measured output rate of f_{ij} during observation interval t, and $\triangle\hat{c}$ is a pre-defined increase parameter. This algorithm is a variant of well-known AIMD.

3.4 Capacity Allocation to Edge-to-Edge Flows

LPS is supposed to allocate the total estimated network capacity C to edge-to-edge flows in such a way that the flows passing through the same bottleneck should share the bottleneck capacity in proportion to their budgets, and also the flows that are not competing with other flows should get all the available capacity on their route. The complicated issue is to do this without knowledge of the topology for network core. We now propose a simple and generic algorithm to perform this centralized rate allocation within Distributed-DCC framework.

First, at LPS, we introduce a new information about each edge-to-edge flow f_{ij}. A flow f_{ij} is *congested*, if egress j has been receiving congestion indications from that flow recently (we will later define what "recent" is).

At LPS, let K_{ij} determine whether f_{ij} is congested or not. If $K_{ij} > 0$, LPS determines f_{ij} as congested. If not, it determines f_{ij} as non-congested. Let's call the time-scale at which LPS and egresses communicate as *LPS interval*. At every LPS interval t, LPS calculates K_{ij} as follows:

$$K_{ij}(t) = \begin{cases} \hat{k},\ f_{ij} \text{ was } \textit{congested} \text{ at } t-1 \\ max(0, K_{ij}(t-1) - 1),\ f_{ij} \text{ was } \textit{non-congested} \text{ at } t-1 \end{cases} \quad (2)$$

where $\hat{k}$ is a positive integer. Notice that $\hat{k}$ parameter defines how long a flow will stay in "congested" state after the last congestion indication. So, $\hat{k}$ defines the time-line to determine if a congestion indication is "recent" or not. Note that instead of setting K_{ij} to $\hat{k}$ at every congestion indication, several different methods can be used for this purpose, but we proceed with the method in (2).

Given the above method to determine whether a flow is congested or not, we now describe the algorithm to allocate capacity to the flows. Let F be the set of all edge-to-edge flows in the diff-serv domain, and F_c be the set of *congested* edge-to-edge flows. Let C_c be the accumulation of $\hat{c}_{ij}$s where $f_{ij} \in F_c$. Further, let B_c be the accumulation of b_{ij}s where $f_{ij} \in F_c$. Then, LPS calculates the allowed capacity for f_{ij} as follows:

$$c_{ij} = \begin{cases} \frac{b_{ij}}{B_c} C_c, \, K_{ij} > 0 \\ \hat{c}_{ij}, \, otherwise \end{cases}$$

The intuition is that if a flow is congested, then it must be competing with other congested flows. So, a congested flow is allowed a capacity in proportion to its budget relative to budgets of all congested flows. Since we assume no knowledge about the interior topology, we *approximate* the situation by considering these congested flows as if they are traversing a single bottleneck. If knowledge about the interior topology is provided, one can easily develop better algorithms by sub-grouping the congested flows that are passing through the same bottleneck. If a flow is not congested, then it is allowed to use its own estimated capacity, which will give enough freedom to utilize capacity available to that flow.

3.5 Fairness

We examine the issues regarding fairness in two main cases:

- *Single-bottleneck case:* The pricing protocol should charge *same price to users of same bottleneck*. In this way, among users of same bottleneck, the ones with more budget will be given more capacity. The intuition behind this reasoning is that the cost of providing capacity to each customer is the same.
- *Multi-bottleneck case:* The pricing protocol should *charge more to users whose traffic traverses more bottlenecks* and causes more costs. So, other than proportionality to user budgets, we also want to allocate less capacity to users whose flows are traversing more bottlenecks than the others. For multi-bottleneck networks, two main types of fairness have been defined: max-min, proportional [10]. In max-min fairness, flows get equal share of bottlenecks, while in proportional fairness flows get penalized according to number of bottlenecks they traverse. Depending on cost structure and user utilities, provider may want to choose max-min or proportional fairness. So, we would like to have ability of tuning pricing protocol such that fairness of its rate allocation is in the way provider wants.

To achieve the above fairness objectives in Distributed-DCC, we introduce new parameters for tuning rate allocation to flows. In order to penalize f_{ij}, egress j can reduce $\hat{b}_{ij}$, which causes decrease in c_{ij}. It uses following function:

$$b_{ij} = f(\hat{b}_{ij}, r(t), \alpha, r_{min}) = \frac{\hat{b}_{ij}}{r_{min} + (r_{ij}(t) - r_{min}) * \alpha}$$

where $r_{ij}(t)$ is the estimated congestion cost caused by f_{ij}, r_{min} is the minimum possible congestion cost, and α is *fairness coefficient*. Instead of $\hat{b}_{ij}$, the egress j now sends b_{ij} to LPS. When α is 0, Distributed-DCC is employing max-min fairness. As it gets larger, the flow gets penalized more and rate allocation gets closer to proportional fairness. However, if it is too large, then the rate allocation will get away from proportional fairness. Let α^* be the α value where the rate allocation is proportionally fair. If the estimation $r_{ij}(t)$ is accurate, then $\alpha^* = 1$.

Assuming that severity of each bottleneck is the same, we can directly use the number of bottlenecks f_{ij} is traversing in order to calculate $r_{ij}(t)$ and r_{min}. In such a case, r_{min} will be 1 and $r_{ij}(t)$ should be number of bottlenecks the flow is passing through. If the interior nodes increment a header field of the packets at the time of congestion, then the egress station can estimate the number of bottlenecks the flow is traversing. We skip description of such an estimation algorithm to keep reader's focus on major issues.

3.6 Scalability

There are mainly two issues regarding scalability: LPS, the number of flows. First of all, the flows are not per-connection basis, i.e. all the traffic going from edge router i to j is counted as only one flow. This actually relieves the scaling of operations happening on per-flow basis. The number of flows in the system will be $n(n-1)$ where n is the number of edge routers in the diff-serv domain. So, indeed, scalability of the flows is not a problem for the current Internet since number of edge routers for a single diff-serv domain is very small.

LPS can be scaled in two ways. First idea is to implement LPS in a fully distributed manner. Edge stations exchange information with each other (like link-state routing). Basically, each station sends total of $n-1$ messages to other stations. So, this will increase overhead on network because of the extra messages, i.e. complexity will increase from $O(n)$ to $O(n^2)$ in terms of number of messages.

Alternatively, LPS can be divided into multiple local LPSs which synchronize among themselves to maintain consistency. This way the complexity of number of messages will reduce. However, this will be at a cost of some optimality again.

4 Edge-to-Edge Pricing Scheme (EEP)

For flow f_{ij}, Distributed-DCC framework provides an allowed capacity c_{ij} and an estimation of total user budget $\hat{b}_{ij}$ at ingress i. So, the provider station at ingress i can use these two information to calculate price. We propose a simple price formula to balance supply c_{ij} and demand $\hat{b}_{ij}$:

$$p_{ij} = \frac{\hat{b}_{ij}}{c_{ij}} \tag{3}$$

Also, the ingress i uses the total estimated network capacity C in calculating the V_{max} contract parameter defined in (1). A simple method for calculating

V_{max} is $V_{max} = C*T$ where T is the contract length. This allows all the available capacity to be contracted by a single flow, which is a loose admission control. More conservative admission control should be used to calculate V_{max}.

We now prove optimality of (3) for a single-bottleneck network. We skip the proof for a multi-bottleneck network for space considerations. We model user i's utility with the well-known [4] [10] function $u_i(x) = w_i log(x)$, where x is bandwidth given to the user and w_i is user i's budget (i.e. willingness-to-pay). Suppose p_i is the price advertised to user i. Then, user i will maximize his surplus S_i by contracting for $x_i = w_i/p_i$.

So, the provider can now figure out what price to advertise to each user by maximizing the social welfare $W = S + R$, where R is the provider revenue. Let $K(x) = kx$ be a linear function and be the cost of providing x amount of capacity to a user, where k is a positive constant. Then social welfare will be:

$$W = \sum_{i=1}^{n} [u_i(x_i) - kx_i]$$

We maximize W subject to $\sum_i x_i \leq C$, where C is the total available capacity. To maximize W, all the available capacity must be given to users since they have strictly increasing utility. Lagrangian and its solution for that system will be:

$$\begin{aligned} W &= \sum_{i=1}^{n} u_i(x_i) - kx_i + \lambda(\sum_{i=1}^{n} x_i - C) \\ x_j &= \frac{w_j}{\sum_{i=1}^{n} w_i} C, \quad j = 1..n \end{aligned} \tag{4}$$

This result shows that welfare maximization of the described system can be done by allocating capacity to the users proportional to their budget, w_i, relative to total user budget. Since the user will contract for $x_i = w_i/p_i$ when advertised a price of p_i, then the optimum price for provider to advertise (i.e. p^*) can be calculated as follows:

$$p^* = p_i = \frac{\sum_{i=1}^{n} w_i}{C}$$

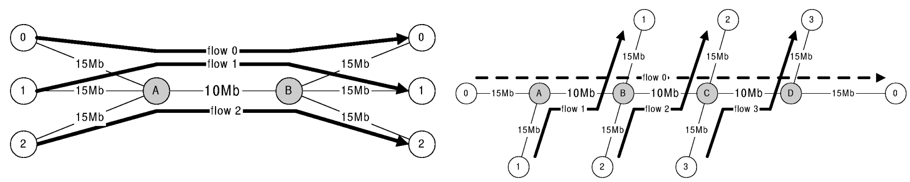
i.e. ratio of total budget to available capacity. So, provider should charge same price to users of same bottleneck route. EEP does that for an edge-to-edge route.

5 Simulation Experiments and Results

We present *ns* [17] simulation experiments of EEP on single-bottleneck and multi-bottleneck topology. Our goals are to illustrate fairness and stability properties of the framework. The single-bottleneck topology is shown in Figure 3-a and the multi-bottleneck topology is shown in Figure 3-b. The white nodes are edge nodes and the gray nodes are interior nodes. To ease understanding

[4] Wang and Schulzrinne introduced a more complex version in [12].

the experiments, each user sends its traffic to a separate egress. For the multi-bottleneck topology, one user sends through all the bottlenecks (i.e. long flow), crossed by the others. Bottleneck links have a capacity of 10Mb/s and all other links have 15Mb/s. Propagation delay on each link is 5ms, and users send UDP traffic with an average packet size of 1000B. Interior nodes mark the packets when local queue exceeds 30 packets. In the multi-bottleneck topology they increment a header field instead of just marking. Buffer size is assumed infinite.



(a) Single-bottleneck network (b) Multi-bottleneck network

Fig. 3. Topologies for Distributed-DCC experiments.

Each user flow maximizes its surplus by contracting for b/p amount of capacity, where b is its budget and p is price. b is randomized according to truncated-Normal distribution with mean b. We will refer to this mean b as flow's budget.

Ingresses send budget estimations to egresses at every *observation interval*. LPS sends information to ingresses at every *LPS interval*. Contracting takes place at every 4s, observation interval is 0.8s, and LPS interval is 0.16s. The parameter $\hat{k}$ is set to 25 (see Section 3.4). $\triangle\hat{c}$ is set to 1 packet (i.e. 1000B), the initial value of $\hat{c}_{ij}$ for each flow f_{ij} is set to 0.1Mb/s, and β is set to 0.95.

5.1 Experiment on Single-bottleneck Topology

We run a simulation experiment for EEP on the single-bottleneck topology, which is represented in Figure 3-a. There are 3 users with budgets of 10, 20, 30 respectively for users 1, 2, 3. Total simulation time is 15000s. Initially, only user 1 is active and after every 5000s one of the other users gets active.

Figure 4-a shows the flow rates averaged over 200 contract periods. We see the flows are sharing the bottleneck capacity almost in proportion to their budgets. The distortion in rate allocation is caused because of the assumptions that the generic edge-to-edge capacity allocation algorithm makes (see Section 3.4). Also, Figure 4-b shows the average prices charged to flows over 200 contract periods. As new users join in, EEP increases the price for balancing supply and demand.

Figure 4-c shows the bottleneck queue size. Notice that queue sizes make peaks transiently at the times when new users gets active. Otherwise, the queue size is controlled reasonably and the system is stable. The reason behind the transient peaks is that the parameter V_{max} is not restricted which causes the newly joining flow to contract for a lot more than the available capacity.

Also, average utilization of the bottleneck link was more than 90%.

5.2 Experiments on Multi-bottleneck Topology

On a multi-bottleneck network, we would like illustrate two properties:

– *Property 1:* provision of various fairness in rate allocation by changing the fairness coefficient α (see Section 3.5)
– *Property 2:* adaptiveness of capacity allocation algorithm (see Section 3.4)

In order to illustrate Property 1, we run a series of experiments for EEP with different α values. We use a larger version of the topology represented in Figure 3-b. In the multi-bottleneck topology there are 10 users and 9 bottleneck links. Total simulation time is 10,000s. Initially, the long flow is active. After each 1000s, one of the other cross flows gets active. So, as the time passes the number of bottlenecks in the system increases. We are interested in the long flow's rate, since it is the one that cause more congestion costs than the other flows.

Figure 4-d shows average rate of the long flow versus number of bottlenecks in the system. As expected, the long flow gets lesser capacity as α increases. When $\alpha = 0$, rate allocation to flows is max-min fair. Observe that when $\alpha = 1$, rate allocation follows the proportionally fair rate allocation. This variation in fairness is basically achieved by advertisement of different prices to the flows. Figure 4-e shows the average price that is advertised to the long flow as number of bottlenecks in the system increases. We can see that the price advertised to the long flow increases as number of bottlenecks increases. As α increases, framework becomes more responsive to the long flow by increasing its price more sharply.

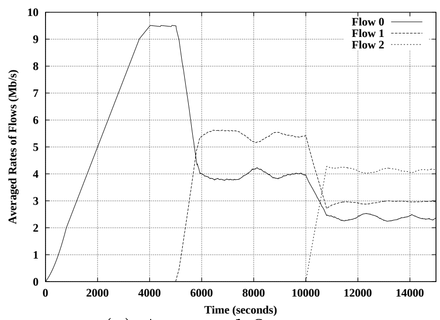
Finally, to illustrate Property 2, we ran an experiment on the topology in Figure 3-b with small changes. We increased capacity of the bottleneck at node D from 10 Mb/s to 15Mb/s. Initially, all the flows have equal budget of 10 units. Total simulation time is 30000s. Between times 10000 and 20000, budget of flow 1 is temporarily increased to 20 units. α is set to 0. All the other parameters are exactly the same as in the single-bottleneck experiments of the previous section.

Figure 4-f shows the given volumes averaged over 200 contracting periods. Until time 10000s, flows 0, 1, and 2 share the bottleneck capacities equally presenting a max-min fair allocation because α is 0. However, flow 3's rate is larger, because bottleneck node D has extra capacity. This is achieved by the freedom given to individual flows by the capacity allocation algorithm (see Section 3.4).

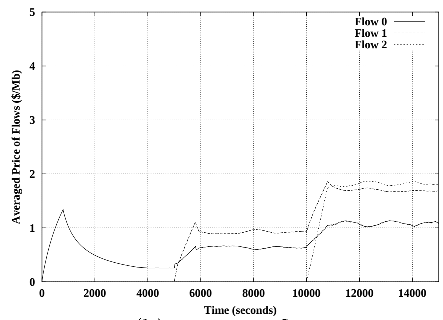
Between times 10000 and 20000, flow 2 gets a step increase in its rate because of the step increase in its budget. In result of this, flow 0 gets a step decrease in its volume. Also, flows 2 and 3 adapt themselves to the new situation, trying to utilize the extra capacity leftover from the reduction in flow 0's rate. So, flows 2 and 3 get a step decrease in their rates. After time 20000, flows restore to their original rates, illustrating adaptiveness of the framework.

6 Summary

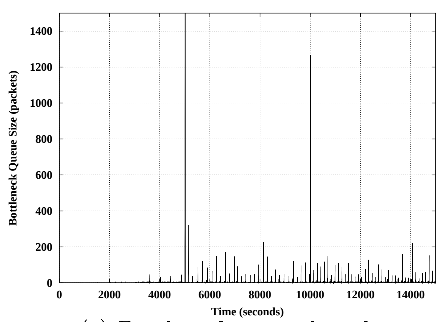
In this paper, we presented a new framework, Distributed-DCC, for congestion pricing in a single diff-serv domain. Distributed-DCC can provide a contracting framework based on *short-term* contracts between multimedia (or any other elastic, adaptive) application and the service provider. Contracting improves QoS if appropriate admission control and provisioning techniques are used. In this paper, we focused on pricing issues.



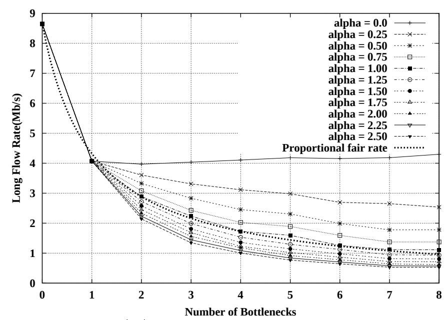
(a) Averaged flow rates



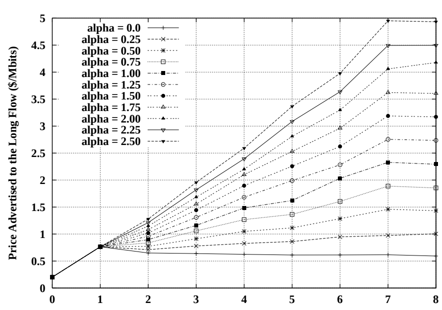
(b) Prices to flows



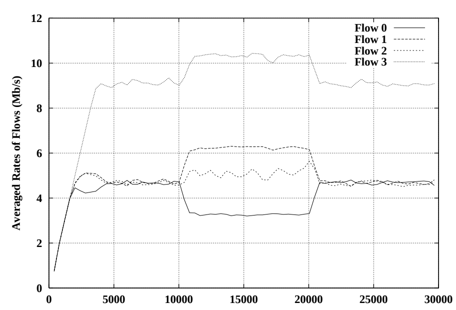
(c) Bottleneck queue length



(d) Rate of long flow



(e) Price to the long flow



(f) Averaged flow rates

Fig. 4. (a)-(c): Results of single-bottleneck experiment for EEP. (d)-(f): Results of EEP experiments on multi-bottleneck topology.

Main contribution of the paper is to develop an *easy-to-implement* congestion pricing architecture which provides flexibility in rate allocation. We investigated fairness issues within Distributed-DCC and illustrated ways of achieving a *range of fairness types* (i.e. from max-min to proportional) through congestion pricing under certain conditions. The fact that it is possible to achieve various fairness types within a single framework is very encouraging. We also developed a pricing scheme, EEP, within the Distributed-DCC framework, and presented several simulation experiments. Future work should include investigation of issues related to Distributed-DCC on multiple diff-serv domains. Also, the framework should be supported by admission control techniques which will tune the contract parameter V_{max} and address minimum QoS settings in SLAs.

References

1. J. K. MacKie-Mason and H. R. Varian, *Pricing the Internet*, Kahin, Brian and Keller, James, 1993.
2. X. Wang and H. Schulzrinne, "An integrated resource negotiation, pricing, and QoS adaptation framework for multimedia applications," *IEEE Journal of Selected Areas in Communications*, vol. 18, 2000.
3. A. Bouch and M. A. Sasse, "Why value is everything?: A user-centered approach to Internet quality of service and pricing," in *Proceedings of IWQoS*, 2001.
4. M. Yuksel, S. Kalyanaraman, and B. Sikdar, "Effect of pricing intervals on congestion-sensitivity of network service prices," Tech. Rep. ECSE-NET-2002-1, Rensselaer Polytechnic Institute, ECSE Networks Lab, 2002.
5. S. Blake et. al, "An architecture for Differentiated Services," *RFC 2475*, 1998.
6. R. Singh, M. Yuksel, S. Kalyanaraman, and T. Ravichandran, "A comparative evaluation of Internet pricing models: Smart market and dynamic capacity contracting," in *Proceedings of Workshop on Information Technologies and Systems (WITS)*, 2000.
7. A. M. Odlyzko, "Internet pricing and history of communications," Tech. Rep., AT & T Labs, 2000.
8. I. Ch. Paschalidis and J. N. Tsitsiklis, "Congestion-dependent pricing of network services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, 2000.
9. A. Gupta, D. O. Stahl, and A. B. Whinston, *Priority pricing of Integrated Services networks*, Eds McKnight and Bailey, MIT Press, 1997.
10. F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *Journal of Operations Research Society*, vol. 49, pp. 237–252, 1998.
11. N. Semret, R. R.-F. Liao, A. T. Campbell, and A. A. Lazar, "Market pricing of differentiated Internet services," in *Proceedings of IWQoS*, 1999, pp. 184–193.
12. X. Wang and H. Schulzrinne, "Pricing network resources for adaptive applications in a Differentiated Services network," in *Proceedings of INFOCOM*, 2001.
13. A. M. Odlyzko, "A modest proposal for preventing Internet congestion," Tech. Rep., AT & T Labs, 1997.
14. D. Clark, *Internet cost allocation and pricing*, Eds McKnight and Bailey, MIT Press, 1997.
15. R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in computer networks: Motivation, formulation and example," *IEEE/ACM Transactions on Networking*, vol. 1, December 1993.
16. M. Yuksel and S. Kalyanaraman, "Simulating the Smart Market pricing scheme on Differentiated Services architecture," in *Proceedings of CNDS part of SCS Western Multi-Conference*, 2001.
17. "UCB/LBLN/VINT network simulator - ns (version 2)," http://www-mash.cs.berkeley.edu/ns, 1997.