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

Murat Yuksel, Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute, Troy, NY
yuksem@cs.rpi.edu, shivkuma@ecse.rpi.edu

Abstract— **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.**

Keywords— **Network Pricing, Congestion Pricing, Quality-of-Service, Fairness, Differentiated-Services**

I. Introduction

Implementation of congestion pricing still remains a challenge, although several proposals have been made, e.g. [1], [2], [3]. Among many others, two major implementation obstacles can be defined: need for *timely feedback* to users about the 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 [4], 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 [5], 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 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 [6], 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 [7], we presented a simple congestion-sensitive pricing framework, *Dynamic Capacity Contracting (DCC)*, for a single diff-serv domain (see Section III). DCC assumed that all the provider stations (that are placed at edge routers) 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 EEP 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 III, we revise overall characteristics of DCC. In Section IV, we develop a simple model for user behavior and make optimization analysis that is basis to our framework, Distributed-DCC. Then, in Section V we describe properties of Distributed-DCC framework, and investigate various issues, such as price calculation, fairness, scalability. Next in Section VI, we develop the pricing scheme EEP. In Section VII, we make experimental comparative evaluation of EEP. We finalize with summary and discussions.

II. 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. [8], [9]), 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 [10], Kelly et al.'s Proportional Fair Pricing (PFP) [11], Semret et al.'s Market Pricing [12], [3], and Wang and Schulzrinne's Resource Negotiation and Pricing (RNAP) [13], [2]. Odlyzko's Paris Metro Pricing (PMP) [14] is an example of static pricing proposal. Clark's Expected Capacity [15] and Cocchi et al.'s Edge Pricing [16] 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 [17], 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

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

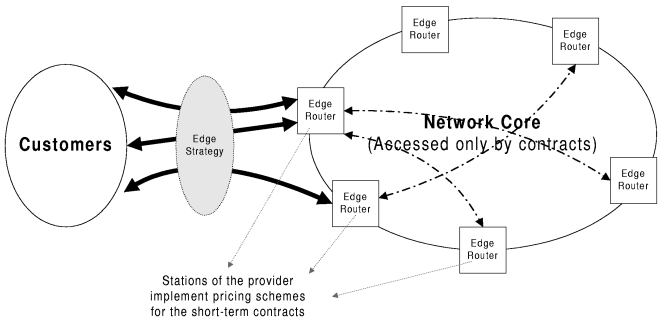


Fig. 1. DCC framework on diff-serv architecture.

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.

III. Dynamic Capacity Contracting (DCC)

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 term of the contract T (length of the contract):

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

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. Access to 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 with the provider for an available contract). So, edge strategy is the decision-making mechanism to identify which customer gets an available contract.

Stations can perfectly advertise congestion-based prices if they have actual information about the congestion level in the network core. This congestion information can come from the interior routers or from the egress edge routers depending on the congestion-detection mechanism being used. DCC assumes that the congestion detection mechanism is able to give congestion information in time scales (i.e. observation intervals) smaller than contracts.

In summary, DCC framework has been designed to use pricing and dynamic capacity contracting as a new dimension in managing congestion, as well as to achieve simple economic goals. The key benefits of DCC are:

- a congestion-sensitive pricing framework employable on diff-serv architecture
- does not require per-packet accounting (works at granularity of contracts)
- does not require upgrades or software support anywhere in the network except the edges

IV. User Adaptation

In this section we present a simple optimization analysis in order to help the reader understand our intuitions behind Distributed-DCC. Please note that the analysis in this section assumes a single bottleneck network and is far from addressing all optimization issues.

We model customer i's utility with the well-known function $u_i(x) = w_i log(x)$ [1] [11], [18], [19], [20], where x is the allocated bandwidth to the customer and w_i is customer i's sensitivity to bandwidth. Then, suppose p_i is the price advertised to a particular user i. The user i will maximize his/her surplus, S_i, by making sure that he/she contracts for $x_i = w_i/p_i$, i.e. :

$$\max_{x_i} S_i = \max_{x_i} \{u_i(x_i) - x_i p_i\}$$

Assuming that the customers obey this above procedure, the provider of the network service 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 the social welfare, W, will be:

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

We maximize W with the condition that $\sum_i x_i = C$, where C is the total available capacity. Notice that to maximize W all the available capacity must be allocated to the users because we assume strictly increasing utility.

Lagrangian and its solution for that system will be as follows:

$$\begin{aligned} W &= \sum_{i=1}^{n} u_i(x_i) - kx_i + \lambda(\sum_{i=1}^{n} x_i - C) \\ \lambda &= k - \frac{\sum_{i=1}^{n} w_i}{C} \\ x_j &= \frac{w_j}{\sum_{i=1}^{n} w_i} C, \quad j = 1..n \end{aligned} \tag{2}$$

This result shows that welfare maximization of the described system can be done only by allocating capacity to the users proportional to their bandwidth sensitivity, w_i, relative to total sensitivity to bandwidth. So, any user i should be given a capacity of

$$x_i = \frac{w_i}{\sum_{i=1}^{n} w_i} C$$

Since we showed that 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:

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

This means that the provider should advertise the same price to all users. We can also interpret user's *budget*, b_i, as his/her

[1] Wang and Schulzrinne introduced a more complex version in [13].

sensitivity to bandwidth, w_i, since a user who is more sensitive to bandwidth is expected to spare more budget for the network service. So, we will use "budget" instead of "sensitivity to bandwidth" for the rest of the paper. Assuming that the customers has a total budget of $B = \sum_i b_i$ for network service per unit time and the network has a capacity of C per unit time, we can rewrite the optimum price as follows:

$$p^* = \frac{B}{C} \tag{3}$$

V. 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) plays a crucial role in terms of functioning of the Distributed-DCC framework. Figure 2 illustrates basic functions (which will be better understood in the following sub-sections) of LPS in the framework.

The following sub-sections investigate and describe several issues[2] regarding the framework.

A. How to Calculate p_{ij} ?

Each ingress station i keeps a "current" price vector p_i, where p_{ij} is the price for the flow from ingress i to egress j. So, the traffic usingflow i to j is charged the price p_{ij}.

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* from ingress i to egress j. The ingresses send their 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 for each particular flow, $\hat{c}_{ij}$. In other words, egress j calculates $\hat{c}_{ij}$ and is informed about $\hat{b}_{ij}$ by ingress i. The egress j, then, penalizes or favors flow i to j by updating its estimated budget value, i.e. $b_{ij} = f(\hat{b}_{ij}, < parameters >)$ where $< parameters >$ are the other parameters that are used for deciding whether to penalize or favor the flow. For example, if the flow i to j is passing through more congested areas than the other flows, the egress j can penalize this flow by reducing its budget estimation $\hat{b}_{ij}$.

At *another time-scale*, the egresses keep sending information to LPS (which can be placed to one of the egresses or can be implemented in a fully distributed manner, see the tech report [21]). More specifically, the egress j sends the following information to LPS:

1. the *updated budget estimations* of all flows passing through itself, i.e. b_{ij} for $i = 1..n$ and $i \neq j$ where n is the number of edge routers

[2]The reader can find more details (e.g. scalability) in our tech report [21].

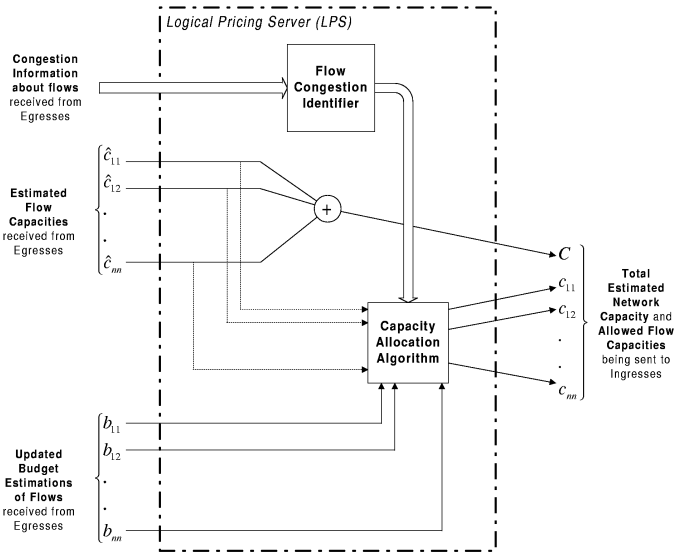


Fig. 2. Major functions of LPS.

2. the *estimated capacities* (please refer to Section V-C for more about capacity estimation) of all flows passing through itself, i.e. $\hat{c}_{ij}$ for $i = 1..n$ and $i \neq j$ where n is the number of edge routers

LPS receives information from egresses and calculates *allowed capacity* c_{ij} for each edge-to-edge flow. Calculation of c_{ij} values is a complicated task which depends on updated budget estimation of each flow (i.e. b_{ij}). 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 $j \neq i$ where n is the number of edge routers

Now, the ingress i calculates price for each flow as follows:

$$p_{ij} = \frac{\hat{b}_{ij}}{c_{ij}} \tag{4}$$

Also, the ingress i uses the total estimated network capacity C in calculating the V_{max} contract parameter defined in (1). Admission control techniques can be used to identify the best value for V_{max}. We use a simple method which does not put any restriction on V_{max}, i.e. $V_{max} = C * T$ where T is the contract length. Notice that we allow flows to contract for more than the available capacity. The available capacity for flow i to j is normally the maximum of link capacities on its route. However, we need to allow flows to contract for more than the available capacity in order to observe their actual demand for capacity. This way we can determine which flow has how much budget (or willingness-to-pay), i.e. demand for capacity. Of course an alternative would be to require users to express their budgets, but indeed users are non-cooperative in this sense. So, as the provider we are supposed to determine user's real incentives for network capacity.

B. 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 the 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 flow i to j in unit time, then the budget estimate for the flow i to j 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.

C. Capacity Estimation at Egresses

The crucial property of capacity estimation in Distributed-DCC is that, it can be made congestion-based, which then makes the prices *congestion-sensitive*. Notice that the price formula in (4) is inversely proportional to the allowed capacity c_{ij}. So, if the network is congested, then c_{ij} will decrease and this will cause price to increase. By similar reasoning, the price will decrease when there is no congestion.

With a simple mechanism (such as marking of packets at interior routers when congested), it is possible to detect congestion at the egress station. So, for a particular edge-to-edge traffic flow, one can make the congestion-based capacity estimation by decreasing the estimation when congestion is detected and by increasing when congestion is not detected for that flow. In this sense, several capacity estimation algorithms can be used, e.g. Additive Increase Additive Decrease (AIAD), Additive Increase Multiplicative Decrease (AIMD). We will provide a full description of such an algorithm later in Section VI.

D. 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:

$$K_{ij}(t) = \begin{cases} \hat{k}, & f_{ij} \text{ was } congested \text{ at } t-1 \\ K_{ij}(t-1) - 1, & f_{ij} \text{ was } non\text{-}congested \text{ at } t-1 \end{cases} \quad (5)$$

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, in other words, $\hat{k}$ 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 (5).

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 can *approximate* the situation by considering these congested flows as if they are passing through a single bottleneck. If knowledge about the interior topology is provided, one can easily develop better algorithms by subgrouping 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 particular flow. The algorithm will be understood more clearly after the simulation experiments in Section VII.

E. Fairness

We examine the issues regarding fairness in two main cases. We first determine these two cases and then provide solutions within Distributed-DCC framework.

E.1 Cases

- *Single-bottleneck case:* The pricing protocol should charge *the same price to the users of the same bottleneck*. In this way, among the customers using the same bottleneck, the ones who have more budget will be given more rate than the others. 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 the customers whose traffic passes through more bottlenecks* and cause more costs to the provider. So, other than proportionality to customer budgets, we also want to allocate less rate to the customers whose flows are passing through more bottlenecks than the other customers.

For multi-bottleneck networks, two main types of fairness have been defined: max-min fairness [18], proportional fairness [11]. In max-min fair rate allocation, all flows get equal share of the bottlenecks, while in proportional fair rate allocation flows get penalized according to the number of traversed bottlenecks. Depending on the cost structure and user's utilities, for some cases the provider may want to choose max-min or proportional rate allocation. The reader can find canonical examples for such cases in [21]. So, we would like to have ability of tuning the

pricing protocol such that fairness of its rate allocation is in the way the provider wants.

E.2 Solutions within Distributed-DCC

In order to achieve the objectives mentioned in the previous section, the pricing framework must give the ability to charge some customers equally while the ability to charge some other customers differently.

To achieve the fairness objectives in Distributed-DCC, we introduce new parameters for tuning rate allocation to flows. In order to penalize flow i to j, the egress j can reduce $\hat{b}_{ij}$ while updating the flow's estimated budget. It uses the following formula to do so:

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

where $r_{ij}(t)$ is the congestion cost caused by the flow i to j, r_{min} is the minimum possible congestion cost for the flow, 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 absolutely correct, then $\alpha^* = 1$. Otherwise, it depends on how accurate $r_{ij}(t)$ is.

Assuming that each bottleneck has the same amount of congestion and capacity. Then, in order to calculate $r_{ij}(t)$ and r_{min}, we can directly use the number of bottlenecks the flow i to j is passing through. 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 (i.e. when its local queue passes a threshold), then at the egress station we can estimate[3] the number of bottlenecks the flow is passing through.

VI. Edge-to-Edge Pricing Scheme (EEP)

One of the main purposes for congestion pricing is to control congestion by making the prices congestion-sensitive. Several studies (e.g. [11]) showed that congestion-sensitive pricing leads to stability. Within the Distributed-DCC framework, there are is one issue to be addressed by a pricing scheme: how to make estimate capacity in a congestion-based manner? In this section, we describe how EEP does that.

In order to make congestion detection at the egress station, we assume that the interior routers mark the packets when their queue passes a threshold. When an egress station receives a marked packet, it treats it as a congestion indication.[4]

Given the above congestion detection mechanism, egress stations make a congestion-based estimation of the capacity for the flows passing through themselves. Remember that estimated capacity, $\hat{c}_{ij}$, for each flow is sent to LPS in Distributed-DCC framework. 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 flow i to j at the end of observation interval t as follows:

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

where β is in (0,1), $\mu_{ij}(t)$ is the measured output rate of flow i to j during observation interval t, and $\triangle\hat{c}$ is a pre-defined increase parameter. This algorithm is a variant of well-known AIMD. Also, notice that the above capacity estimation algorithm is congestion-based as it is necessary for the congestion-sensitivity of Distributed-DCC framework (see Section V-C). So, egresses make capacity estimation for each flow according to the above algorithm, and send $\hat{c}_{ij}(t)$ as the current estimated capacity for flow i to j.

VII. Simulation Experiments and Results

We now present *ns* [22] simulation experiments of EEP on single-bottleneck and multi-bottleneck topology. Our goals are to illustrate fairness and stability properties of the scheme.

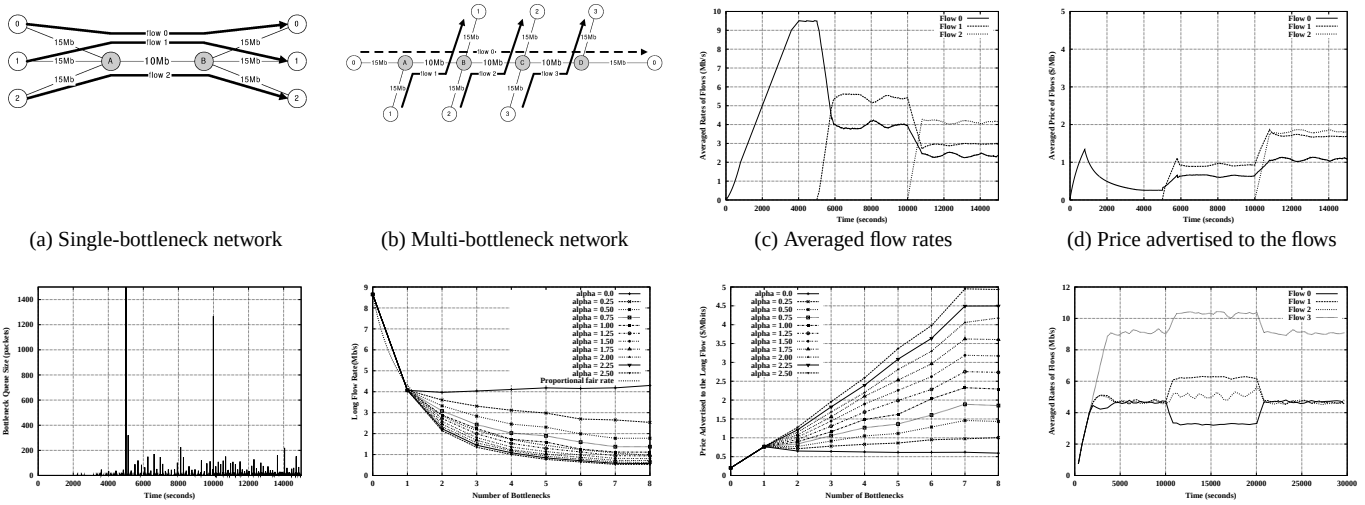
The single-bottleneck topology has a bottleneck link, which is connected to n edge nodes at each side where n is the number of users. The multi-bottleneck topology has $n-1$ bottleneck links, that are connected to each other serially. There are again n ingress and n egress edge nodes. Each ingress edge node is mutually connected to the beginning of a bottleneck link, and each egress node is mutually connected to the end of a bottleneck link. All 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. To ease understanding 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) while the others cross that user's long flow. The queues at the interior nodes (i.e. nodes that stand at the tips of bottleneck links) mark the packets when their local queue size exceeds 30 packets. Buffer size is assumed to be infinite. In the multi-bottleneck topology they increment a header field instead of just marking. Figure 3-a shows a single-bottleneck topology with $n = 3$. Figure 3-b shows multi-bottleneck topology with $n = 4$. The white nodes are edge nodes and the gray nodes are interior nodes. These figures also show the traffic flow of users on the topology.

The user flow tries to maximize its surplus by contracting for b/p amount of capacity, where b is its budget and p is price. The flows's budgets are randomized according to Normal distribution with a given mean value. This mean value is what we will refer to as flows's budget in our simulation experiments.

Ingresses send budget estimations to corresponding 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, which means a flow is determined to

[3] Description of a full algorithm for that estimation is available in [21].

[4] Notice that this is only one particular way of detecting congestion. Distributed-DCC does not necessarily need the interior routers to mark packets, as long as other ways of detecting congestion are available.



(a) Single-bottleneck network (b) Multi-bottleneck network (c) Averaged flow rates (d) Price advertised to the flows

(e) Bottleneck queue length (f) Rate of long flow (g) Price advertised to the long flow (h) Averaged flow rates

Fig. 3. (a)-(b): Topologies for Distributed-DCC experiments. (c)-(e): Results of single-bottleneck experiment for EEP. (f)-(h): Results of EEP experiments on multi-bottleneck topology.

be non-congested at least after (please see Section V-D) 25 LPS intervals equivalent to one contracting interval.

The parameter $\triangle\hat{c}$ 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.

A. Experiment on Single-bottleneck Topology

We run simulation an experiment for EEP on the single-bottleneck topology, which is represented in Figure 3-a. In this experiment, there are 3 users with budgets of 10, 20, 30 respectively for users 1, 2, 3. Total simulation time is 15000s, and at the beginning only the user 1 is active in the system. After 5000s, the user 2 gets active. Again after 5000s at simulation time 10000, the user 3 gets active.

In terms of results, each flow's rate is very important. Figure 3-c 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 V-D).

Figure 3-d shows the price being advertised to flows. As the new users join in, EEP increases the price in order to balance supply and demand. Also, we can see the same dynamic as in the volume allocation graphs caused by the capacity allocation algorithm.

Figure 3-e 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.

During the simulation, average utilization of the bottleneck link was more than 90%, and no packet drops were allowed.

B. Experiments on Multi-bottleneck Topology

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

- *Property 1:* provision of various fairness in rate allocation by changing the fairness coefficient α of Distributed-DCC framework (see Section V-E.2)
- *Property 2:* performance of the capacity allocation algorithm in terms of adaptiveness (see Section V-D)

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. At the beginning, the user with the long flow is active. After each 1000s, one of these other users gets active. So, as the time passes the number of bottlenecks in the system increases since new users with crossing flows join in. We are interested in the rate of the long flow, since it is the one that cause more congestion costs than the other user flows.

Figure 3-f shows the average rate of the long flow versus the number of bottlenecks in the system. As expected the long flow gets less and less capacity as α increases. When $\alpha = 0$, the scheme achieves max-min fairness. Observe that when $\alpha = 1$, rate allocation goes along with proportionally fair rate allocation. This variation in fairness is basically achieved by advertisement of different prices to the user flows. Figure 3-g shows the average price that is advertised to the long flow as the number of bottlenecks in the system increases. We can see that the price advertised to the long flow increases as the number of bottlenecks increases. As α increases, the scheme 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. There are four flows and three bottlenecks in the network as represented in Figure 3-b. Initially, all the flows have an equal budget of 10. Total simulation time is 30000s. Between times 10000 and 20000, budget of flow 1 is temporarily increased to 20. The fairness coefficient α is set to 0. All the other parameters are exactly the same as in the single-bottleneck experiments of the previous section.

Figure 3-h 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 was set to 0. However, flow 3 is getting more than the others because of the extra capacity at bottleneck node D. This flexibility is achieved by the freedom given to individual flows by the capacity allocation algorithm (see Section V-D).

Between times 10000 and 20000, flow 2 gets a step increase in its allocated volume 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 by attempting to utilize the extra capacity leftover from the reduction in flow 0's volume. So, flow 2 and 3 gets a step decrease in their volumes. After time 20000, flows restore to their original volume allocations, illustrating the adaptiveness of the scheme.

VIII. Summary

In this paper, we presented a new framework, Distributed-DCC, for congestion pricing in a single diff-serv domain. 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 severeal simulation experiments.

Future work should include investigation of issues related to extending Distributed-DCC on multiple diff-serv domains. Another future work item is to implement soft admission control techniques in the framework by tuning the contract parameter V_{max}. Currently, V_{max} is set to total network capacity, which allows individual users to contract for significantly larger than the network can handle. Several other improvements are possible to the framework such as better capacity estimation techniques (see Section V-C), better budget estimation techniques (see Section V-B).

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, "RNAP: A resource negotiation and pricing protocol," in *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 1999, pp. 77–93.

[3] 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," *IEEE Journal on Selected Areas of Communications – to be published*, 2001.

[4] A. Bouch and M. A. Sasse, "Why value is everything?: A user-centered approach to Internet quality of service and pricing," in *Proceedings of IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, 2001.

[5] M. Yuksel, S. Kalyanaraman, and B. Sikdar, "Effect of pricing intervals on the congestion-sensitivity of network service prices," in *Submitted to GLOBECOM*, 2002.

[6] S. Blake et. al, "An architecture for Differentiated Services," *IETF RFC 2475*, December 1998.

[7] 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.

[8] A. M. Odlyzko, "Internet pricing and history of communications," Tech. Rep., AT & T Labs, 2000.

[9] I. Ch. Paschalidis and J. N. Tsitsiklis, "Congestion-dependent pricing of network services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 171–184, 2000.

[10] A. Gupta, D. O. Stahl, and A. B. Whinston, *Priority pricing of Integrated Services networks,* Eds McKnight and Bailey, MIT Press, 1997.

[11] 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.

[12] N. Semret, R. R.-F. Liao, A. T. Campbell, and A. A. Lazar, "Market pricing of differentiated Internet services," in *Proceedings of IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, 1999, pp. 184–193.

[13] X. Wang and H. Schulzrinne, "Pricing network resources for adaptive applications in a Differentiated Services network," in *Proceedings of Conference on Computer Communications (INFOCOM)*, 2001.

[14] A. M. Odlyzko, "A modest proposal for preventing Internet congestion," Tech. Rep., AT & T Labs, 1997.

[15] D. Clark, *Internet cost allocation and pricing,* Eds McKnight and Bailey, MIT Press, 1997.

[16] 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.

[17] M. Yuksel and S. Kalyanaraman, "Simulating the Smart Market pricing scheme on Differentiated Services architecture," in *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS) part of Western Multi-Conference*, 2001.

[18] S. Kunniyur and R. Srikant, "End-to-end congestion contro: Utility functions, random losses and ecn marks," in *Proceedings of Conference on Computer Communications (INFOCOM)*, 2000.

[19] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, October 2000.

[20] S. H. Low and D. E. Lapsley, "Optimization flow control – I: Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–875, 1999.

[21] M. Yuksel and S. Kalyanaraman, "Distributed dynamic capacity contracting: An overlay congestion pricing framework for diff-serv," Tech. Rep., Rensselaer Polytechnic Institute, Available at http://networks.ecse.rpi.edu/~ yuksem/full-ddcc.pdf, 2001.

[22] "UCB/LBLN/VINT network simulator - ns (version 2)," http://www-mash.cs.berkeley.edu/ns, 1997.