Distributed Dynamic Capacity Contracting: An overlay congestion pricing framework for differentiated-services architecture

Murat Yuksel, Shivkumar Kalyanaraman, Anuj Goel
Rensselaer Polytechnic Institute, Troy, NY
yuksem@cs.rpi.edu, shivkuma@ecse.rpi.edu, goela@cs.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 two pricing schemes based on the manner of using pricing to control congestion: Pricing for Congestion Control (PFCC) and Pricing over Congestion Control (POCC). PFCC uses pricing directly for controlling congestion, whilst POCC uses an underlying edge-to-edge congestion control mechanism by overlaying pricing on top of it.**

Keywords— **Network Pricing, Congestion Pricing, Quality-of-Service, Fairness, Congestion Control, 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 wide area 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). This means that in order to achieve congestion control through pricing, service prices must be updated very frequently (i.e. 2-3 seconds since RTT is expressed in terms of milliseconds for most cases in the Internet). 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.

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

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 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.

DCC 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 two pricing schemes based on the way of approaching congestion control problem: Pricing *for* Congestion Control (PFCC), Pricing *over* Congestion Control (POCC). While PFCC tries to control congestion directly by pricing, POCC overlays pricing on top of an existing edge-to-edge congestion control mechanism. We illustrate stability of the schemes by simulation experiments. We address fairness problems related to pricing, and show that PFCC can achieve max-min and proportional fairness by tuning a parameter, called as *fairness coefficient*.

The rest of 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) regarding it. Next, in Section VI, we develop two pricing schemes (PFCC and POCC) based on method of using pricing to control congestion. In Section VII, we make experimental comparative evaluation of PFCC and POCC. We finalize with summary and discussions.

II. Related Work

There has been several pricing proposals, which can be classified in many ways: *static* vs. *dynamic*, *per-packet* charging vs. *per-contract* charging, and charging *prior* to service vs. *posterior* to service.

Although there are opponents to dynamic pricing in the area (e.g. [8], [9], [10]), 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 [11], Kelly et al.'s Proportional Fair Pricing (PFP) [12], Semret et al.'s Market Pricing [13], [3], and Wang and Schulzrinne's Resource Negotiation and Pricing (RNAP) [14], [2]. Odlyzko's Paris Metro Pricing (PMP) [15] is an example of static pricing proposal. Clark's Expected Capacity [16] and Cocchi et al.'s Edge Pricing [17] 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 (i.e. excessive overhead) and its many requirements from routers (e.g. requires all routers to be updated). In [18], 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. We believe that pricing schemes that require upgrades to all routers will eventually fail in implementation phase. This is because of the fact that the Internet routers are owned by different entities who may or may not be willing to cooperate in the process of router upgrades. 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):

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

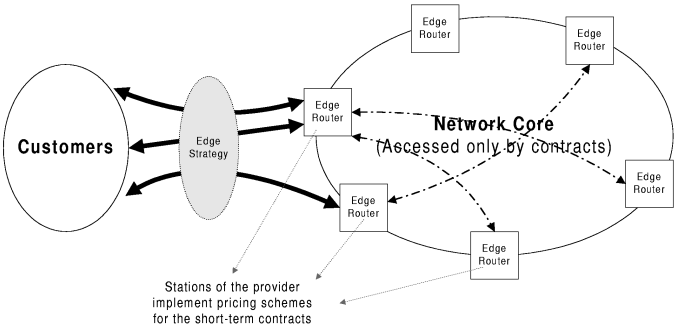


Fig. 1. DCC framework on diff-serv architecture.

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 at the provider station.

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 that is basis to Distributed-DCC. One important characteristic of congestion-sensitive pricing is that the price must be oscillating around an optimum price, p^*, to guarantee both congestion control and high utilization of network resources. We now derive a formula for p^* by explicitly modeling customer utilities. We model customer i's utility with the well-known function $u_i(x) = w_i log(x)$ [1] [12], [19], [20], [21], 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

[1] Wang and Schulzrinne introduced a more complex version of this function in [14].

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. :

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

$$\frac{dS_i}{dx_i} = \frac{w_i}{x_i} - p_i = 0$$

$$x_i = \frac{w_i}{p_i}$$

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:

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

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

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 functions (i.e. $log(x)$) for them.

By applying Lagrange-Multiplier Method [22], we first convert W to the following:

$$W \equiv Z = \sum_{i=1}^{n} u_i(x_i) - kx_i + \lambda(\sum_{i=1}^{n} x_i - C)$$

We can get the following system of equations by equating partial derivative of W to zero for each unknown variable:

$$\begin{aligned} Z_\lambda &= \sum_{i=1}^{n} x_i - C = 0 \\ Z_{x_j} &= \frac{w_j}{x_j} - k + \lambda = 0, \quad j = 1..n \end{aligned} \tag{2}$$

After solving system of equations 2, we get the solution as follows:

$$\begin{aligned} \lambda &= k - \frac{\sum_{i=1}^{n} w_i}{C} \\ x_j &= \frac{w_j}{\sum_{i=1}^{n} w_i} C, \quad j = 1..n \end{aligned} \tag{3}$$

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 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:

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

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

This means that the provider should advertise the same price to all users. However, notice that this above study assumed two major things:

- the cost for provisioning capacity per unit bandwidth to each user is the same
- all users have the same type of utility function, i.e. $u(x) = w\ log(x)$

Since optimality is not our single goal in pricing, we do not focus on addressing the above assumptions. Study of how much optimality can be achieved by Distributed-DCC is left for future work.

We can also interpret user's *budget*, b_i, as his/her *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:

$$p^* = \frac{B}{C} \tag{4}$$

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. In Distributed-DCC framework, 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 contracting takes place between the customers and the ingress stations. So, each ingress station keeps a "current" price value for each edge-to-edge flow and advertises that price value, p_{ij}, to customers of the flow from ingress i to egress j.

A. How to Calculate 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

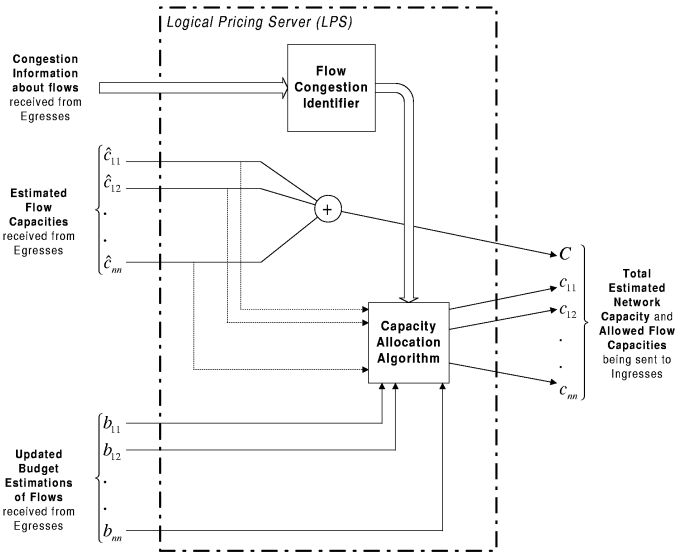


Fig. 2. Major functions of LPS.

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 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, then 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 Section V-E). 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 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 get 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 can calculate price for each flow as follows:

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

Also, the ingress i can use the total estimated network capacity C in calculating the V_{max} contract parameter defined in Equation 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.

One can claim: "Why not use c_{ij} for calculating V_{max}?" This will prevent flows to send more than available capacity. However, it will not allow flows to compete for the capacity, which will not give opportunity to the flows with higher budget than the others. By allowing flows to contract for more than the allowed, we can leave the sharing of the total available capacity to the flows themselves. In this way, the flows will share the capacity based on their budgets, i.e. willingness to pay.

B. Budget Estimation at Ingresses

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 (this is a realistic assumption) 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 $\hat{b}_{ij} = x_{ij}p_{ij}$.

C. Capacity Estimation at Egresses

The crucial property of capacity estimation in Distributed-DCC is that, it can be made congestion-based. 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 capacity estimation congestion-sensitive 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-A.

D. Fairness

Fairness of a pricing scheme is indirectly equivalent to fairness of rate allocation schemes. A pricing scheme determines and advertises the prices to the users, then the users contract for network capacity according to their budgets. So, a pricing scheme allocates network capacity indirectly.

Assuming that users have a utility function of form $u(x) = w\ log(x)$, in Section IV, we provided a simple proof that the provider should advertise a price of $p = B/C$, where B is the total budget of the users and C is the network capacity, in order to maximize social welfare. This will cause users to share the network capacity in proportion to their budgets. However, fairness issues are not that simple. We examine the issues regarding fairness in two main cases. We now first determine these two cases and then provide solutions within Distributed-DCC framework.

D.1 Cases

1. *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 willingness to pay 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. The simple proof in Section IV also shows that charging all users equally is optimal in such a case.
2. *Multi-bottleneck case:* The pricing protocol should charge more to the customers who cause more costs (e.g. congestion) to the provider. This will make sure that the revenue will be higher while being fairer to the customers. 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 [19], proportional fairness [12]. We now study proportional fairness and max-min fairness in terms of social welfare maximization with a canonical example. Consider a multi-bottleneck network in which there is a long flow that is crossed by n parallel flows. An example of such a network is shown in Figure 4-b. Suppose all the bottlenecks are equivalent in capacity, C. Intuitively, whatever the long flow gets, all the parallel flows will get the rest of the capacity. Let x_0 be the capacity given to the long flow and x_1 be the capacity given to one of the parallel flows. Suppose that the utility of the long flow is $u_0(x_0) = w_0 log(x_0)$ and the utility of one of the parallel flows is $u_1(x_1) = w_1 log(x_1)$. Notice that w_0 and w_1 are the sensitivity of the flows to capacity (also interpreted as flow's budget, see Section IV). Since the long flow is passing through n bottlenecks, cost of providing capacity to the long flow is n times more than cost of providing capacity to one of the parallel flows. So, let cost of providing x_1 to one of the parallel flows be $K_1(x_1) = kx_1$, and let the cost of providing x_0 to the long flow be $K_0(x_0) = nkx_0$. Within this context, the social welfare, W, and its Lagrangian will be:

$$W = w_0 log(x_0) + nw_1 log(x_1) - nkx_0 - nkx_1$$

$$W \equiv Z = w_0 log(x_0) + nw_1 log(x_1) - nkx_0 - nkx_1 + \lambda(x_0 + x_1 - C)$$

After solving the above Lagrangian, we get the following solutions for x_0 and x_1 to maximize W:

$$x_0 = \frac{w_0 C}{w_0 + nw_1}$$

$$x_1 = \frac{nw_1 C}{w_0 + nw_1}$$

From the above result, we make two observations:

- First, if both the long flow and a parallel flow have equal bandwidth sensitivity, i.e. $w_0 = w_1$, then the optimal allocation will be $x_0 = C/(n+1)$ and $x_1 = Cn/(n+1)$. This is the *proportional fair* case. So, proportional fairness is optimal only when all the flows have equal bandwidth sensitivity. As another interpretation, it is optimal only if all the flows have equal budget.
- Second, if the long flow is sensitive to bandwidth n times more than a parallel flow, i.e. $w_0 = nw_1$, then the optimal allocation will be $x_0 = x_1 = C/2$. This is the *max-min fair* case. So, max-min fairness is optimal only when the long flow's utility is sensitive to bandwidth in proportion to the cost of providing capacity to it. In other words, by interpreting bandwidth sensitivity as the flow's budget, max-min fairness is optimal only when the long flow has budget in proportion to the cost of providing capacity to it.

Observations similar to above have been made in the area, e.g. [12], [23].

D.2 Solutions within Distributed-DCC

In order to achieve the objectives mentioned in the previous section, the pricing protocol should charge the customers equally in a single-bottleneck topology, while it should charge them differently in a multi-bottleneck topology. This means that 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. α can be 0 at minimum. When it is greater than 0, it means that Distributed-DCC is employing proportional fairness in its rate allocation. When it is 0, Distributed-DCC is employing max-min fairness. So, using larger α values yield to more proportional fairness in the rate allocation of Distributed-DCC.

Assuming that each bottleneck has the same amount of congestion and also assume that they have the same 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, then at the egress station we can estimate the number of bottlenecks the flow is passing through. To do that estimation, the egress station does the following at time interval t for flow i to j:

$$r_{ij}(t) = \begin{cases} \hat{r}_{ij}(t), & r_{ij}(t-1) \leq \hat{r}_{ij}(t) \\ r_{ij}(t-1) - \triangle r, & otherwise \end{cases} \quad (5)$$

where $\hat{r}_{ij}(t)$ is the highest number of bottlenecks that flow passed through in time interval t, $\triangle r$ is a pre-defined value. Realize that the header field of the packets are being incremented only if they are passing through a congested bottleneck. It is possible that some of the bottlenecks are not congested when a particular packet is passing through them. For example, the header field of the packet may be incremented only three times, although it actually passed through six bottlenecks. So, it is necessary to bias the estimation to the largest number of bottlenecks

the packets of that flow have passed recently. Also as another issue, IP routing causes route of the flows to change dynamically. To consider the dynamic behavior of the routes, it is also necessary to decrease r_{ij} when $r_{ij}(t-1) > \hat{r}_{ij}(t)$. So, if the route of the flow has changed, then after some time (depending on how large the $\triangle r$ is) the value of r_{ij} will decrease to the actual number of bottlenecks the flow is passing through.

E. Scalability

Distributed-DCC operates on per edge-to-edge flow basis. 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 scalability problem for operations that happen 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. If it becomes so large in future, then aggregation techniques can be used to overcome this scalability issue, of course, by sacrificing some optimality.

Scalability of LPS can be done two ways. First idea is to implement LPS in a fully distributed manner. The edge stations exchange information with each other (similar to link-state routing). Basically, each station will send total of $n-1$ messages, each of which headed to other stations. So, this will increase the overhead on the network because of the extra messages, i.e. the 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.

Since these above-defined scaling techniques are very well-known, we do not focus on detailed description of them.

VI. Distributed-DCC: Pricing Schemes

One of the main purposes for congestion pricing is to control congestion by making the prices congestion-sensitive. Several studies (e.g. [12], [24]) showed that congestion-sensitive pricing leads to stability. Within Distributed-DCC framework, we now describe two pricing schemes which are mainly inspired of different approaches to the problem of congestion control. *Pricing for Congestion Control* (PFCC) uses pricing directly for the purpose of congestion control, while *Pricing over Congestion Control* (POCC) uses an underlying edge-to-edge congestion control mechanism to impose tighter control on congestion. Figure 3 illustrates the big picture of the two approaches. In the following sub-sections, we will describe and investigate these two approaches within Distributed-DCC framework.

A. PFCC

A.1 Capacity Estimation and Congestion Detection

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.[2]

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 and is introduced in [25]. 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.

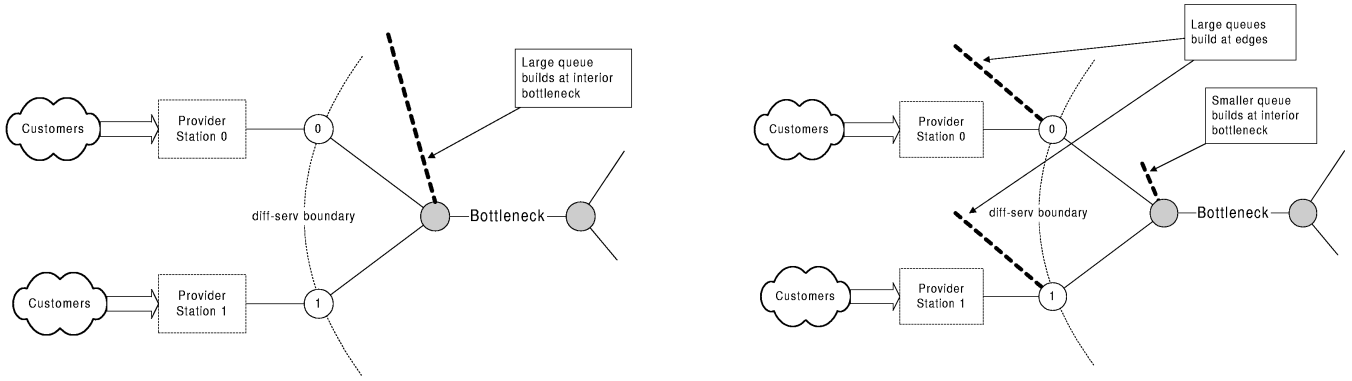
As a refinement to the scheme, the value of the $\triangle\hat{c}$ should decrease when the number of active flows increases. The reason is that, at LPS, the total estimated network capacity C is the accumulation of $\hat{c}_{ij}$s sent from egresses for each flow. So, for example, let there be 3 flows that are outputting some traffic. Each of the corresponding egresses will basically increase their $\hat{c}_{ij}$s by $\triangle\hat{c}$ when there was no congestion for that flow at the last interval. However, when those individual $\hat{c}_{ij}$s are accumulated at LPS, the overall network capacity estimation, C, will increase by $3*\triangle\hat{c}$. This basically shows that the fidelity of control will be dependent on the total number of active flows. Hence, in order to achieve the same level of control fidelity, the $\triangle\hat{c}$ parameter that is being used locally at each egress must be reduced when more flows get active.

Let δ be the best value for $\triangle\hat{c}$ parameter when there is only one flow that is outputting traffic. Then the general formula will be

$$\triangle\hat{c} = \delta/N$$

where N is the number of currently active flows. In Distributed-DCC framework, we can calculate N. The egresses count a flow as "active" if $\mu_{ij} > 0$. They, then, convey the number of active flows to LPS, which later on informs them back with the total number of active flows, N, in the network. This is an optional refinement to the capacity estimation capability of PFCC. We will use this refinement in our simulation experiments later in Section VII.

[2]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) (b)

Fig. 3. (a) PFCC: Pricing with no underlying edge-to-edge congestion control. (b) POCC: Pricing over edge-to-edge congestion control.

A.2 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). Egress j knows whether each flow f_{ij} was congested in the last observation interval or not (please see the previous section). To determine if f_{ij} is currently congested or not, define $k_{ij}(t)$ at egress j as follows:

$$k_{ij}(t) = \begin{cases} 1, & f_{ij} \text{ was } congested \text{ in observation } t-1 \\ 0, & f_{ij} \text{ was } non\text{-}congested \text{ in observation } t-1 \end{cases}$$

To determine whether f_{ij} is congested or not at LPS, egress j sends the current value of k_{ij} along with the other information (i.e. b_{ij} and $\hat{c}_{ij}$) to LPS. Of course, this is done for all flows simultaneously.

At LPS, let K_{ij} be the maintained parameter in order to 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. LPS regularly receives messages from egresses. Let's call the time interval between these messages as *LPS interval*. At every LPS interval t, LPS updates K_{ij} as follows:

$$K_{ij}(t) = \begin{cases} \hat{k}, & k_{ij}(t) = 1 \\ K_{ij}(t-1) - 1, & k_{ij}(t) = 0 \end{cases} \quad (6)$$

where $\hat{k}$ is a positive integer. Notice that $\hat{k}$ parameter defines long a flow can 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 (6).

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:

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

So, a congested flow competes with other congested flows and is allowed a capacity in proportion to its budget relative to budgets of all congested flows. 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.

B. POCC

The essence of POCC is to overlay pricing on top of congestion control, which is a novel approach. Assuming that there is an underlying edge-to-edge congestion control scheme, we can set the parameters of that underlying scheme such that it leads to fairness and better control of congestion. The pricing scheme on top can determine user incentives and set the parameters of the underlying edge-to-edge congestion control scheme accordingly. This way, it will be possible to favor some traffic flows with higher willingness-to-pay than the others. Furthermore, the pricing scheme will also bring benefits such as an indirect control on user demand by price, which will in turn help the underlying edge-to-edge congestion control scheme to operate more smoothly. However the overall system performance (e.g. fairness, utilization, throughput) will be dependent on the flexibility of the underlying congestion control mechanism.

We now describe a possible POCC scheme by overlaying PFCC (which is a purely pricing scheme) on top of an edge-to-edge congestion control scheme Riviera [25]. We will first

provide a general description of Riviera, then outline problems revealed by overlaying PFCC over Riviera. We will complete the description of POCC by providing solutions to the outlined problems. The simulation experiments in Section VII will use this description of POCC.

B.1 Edge-to-Edge Congestion Control: Riviera

Riviera takes advantage of two-way communication between ingress and egress edge routers in a diff-serv network. Ingress sends a *forward* feedback to egress in response to feedback from egress, and egress sends *backward* feedback to ingress in response to feedback from ingress. So, ingress and egress of a traffic flow keep bouncing feedback to each other. Ignoring loss of data packets, the egress of a traffic flow measures the accumulation, a, caused by the flow by using the bounced feedbacks and RTT estimations.

The egress node keeps two threshold parameters to detect congestion: max_thresh and min_thresh. For each flow, the egress keeps a variable that says whether the flow is congested or not. When a for a particular flow exceeds max_thresh, the egress updates the variable to *congested*. Similarly, when a is less than min_thresh, it updates the variable to *not-congested*. It does not update the variable if a is in between max_thresh and min_thresh. The ingress node gets informed about the congestion detection by backward feedbacks and employs AIMD-ER (i.e. a variant of regular AIMD) to adjust the sending rate.

In a single-bottleneck network, Riviera can be tuned such that each flow gets weighted share of the bottleneck capacity. The ingress nodes maintain an additive increase parameter, α, and a multiplicative decrease parameter, β, for each edge-to-edge flow. These parameters are used in AIMD-ER. Among the edge-to-edge flows, by setting the increase parameters (α) at the ingresses and the threshold parameters (max_thresh and min_thresh) at the egresses in ratio of desired rate allocation, it is possible to make sure that the flows get the desired rate allocation. For example, assume there are two flows 1 and 2 competing for a bottleneck (similar to Figure 4-a). If we want flow 1 to get a capacity of w times more than flow 2, then the following conditions must be hold:

1. $\alpha_2 = w\ \alpha_1$
2. $max\_thresh_2 = w\ max\_thresh_1$
3. $min\_thresh_2 = w\ min\_thresh_1$

B.2 POCC: Problems

Overlaying PFCC over Riviera raises two major problems[3]:

1. *Parameter mapping:* Since PFCC wants to allocate network capacity according to the users' budgets that changes dynamically over time, it is a required ability set corresponding parameters of Riviera such that it allocates the capacity to the user flows according to their budgets. So, this raises need for a method of mapping PFCC parameters to the Riviera parameters. Notice that this type of mapping requires Riviera to be able to provide parameters that tunes the rate being given to the edge-to-edge flows.

[3]Note that these problems are not specific to PFCC over Riviera. They are general problems for overlaying a solely pricing scheme over an edge-to-edge congestion control mechanism.

2. *Edge queues:* The underlying congestion control scheme Riviera will not always allow all the traffic admitted by the pricing scheme PFCC, which will cause queues to build up at the network edges. So, management of these edge queues is necessary to overlay PFCC over Riviera. Figures 3-a and 3-b compare the situation of the edge queues in the two cases when there is an underlying congestion control scheme and when there is not.

B.3 POCC: Solutions

1. *Parameter mapping:* For each edge-to-edge flow, LPS can calculate the capacity share of that flow out of the total network capacity. Let $\gamma_{ij} = c_{ij}/C$ be the fraction of network capacity that must be given to the flow i to j. Along with c_{ij}s, LPS can convey γ_{ij}s to the ingress stations, and they can multiply the increase parameter α_{ij} with γ_{ij}. Also, LPS can send γ_{ij}s to the egresses, and they can multiply max_thresh_{ij} and min_thresh_{ij} with γ_{ij}. This solves the parameter mapping problem defined in the previous section.

2. *Edge queues:* We now propose solutions to the second problem, i.e. management of edge queues. Each ingress station i can manage the edge queue with the already available information:

- c_{ij}, allowed capacity of flow i to j (in packet/seconds)
- Q_{ij}, current size of edge queue for flow i to j (in packets) [4]
- T, contract length (in seconds)

Notice that ingress stations get the allowed capacities for flows from LPS, and then calculate prices according to that information. So, one intuitive way of making sure that the user will not contract for more than the amount that the network can handle is to subtract necessary capacity to drain the already built edge queue from c_{ij}, and then make contracts accordingly. In other words, the ingress station re-calculates the allowed capacity for flow i to j by the following formula $c'_{ij} = c_{ij} - Q_{ij}/T$, and uses c'_{ij} for calculating price, i.e. $p_{ij} = \hat{b}_{ij}/c'_{ij}$.

Distributed-DCC can also employ another technique to manage the edge queues. Remember that PFCC makes capacity estimation at egresses according to marking of packets. Specifically, marked packet is counted as a congestion indication and capacity estimation is reduced, which in turn causes price for that flow to increase. So, the ingress station can mark the packets if size of the edge queue exceeds a threshold. This will indirectly reduce the capacity estimation, and hence drain the edge queue. Notice that it is also possible to employ this method simultaneously with the method described in the previous paragraph. In the simulation experiments of the next section we will use both techniques simultaneously. We are working on more conservative approaches by making more pessimistic capacity estimations, in order to manage the edge queues.

VII. Simulation Experiments and Results

We now present *ns* [26] simulation experiments of PFCC and POCC on single-bottleneck and multi-bottleneck topology. Our goals are to illustrate fairness and stability properties of the two schemes with possible comparisons of two.

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

[4]Notice that the edge queue at ingress i is $Q_i = \sum_{j \neq i} Q_{ij}$.

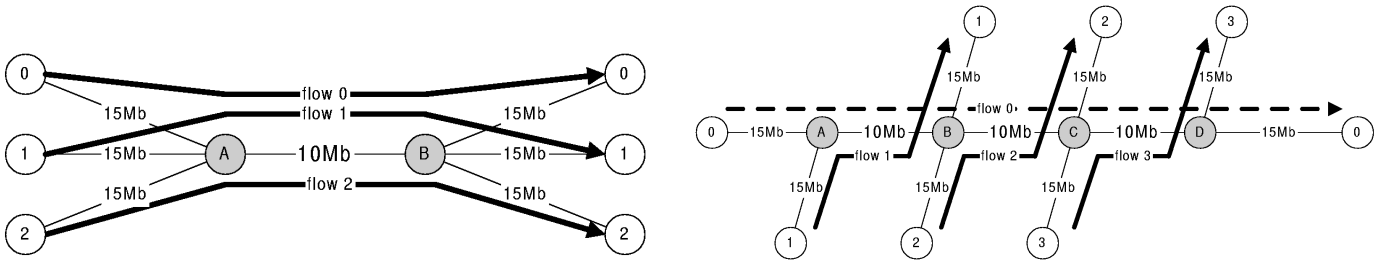


Fig. 4. (a) Single-bottleneck (b) Multi-bottleneck network for Distributed-DCC experiments.

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. In the multi-bottleneck topology they increment a header field instead of just marking. Figure 4-a shows a single-bottleneck topology with $n = 3$. Figure 4-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 total utility 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.

Contracting takes place at every 4s, observation interval is 0.8s, and LPS interval is 0.16s. Ingresses send budget estimations to corresponding egresses at every observation interval. LPS sends information to ingresses at every LPS interval. The parameter $\hat{k}$ is set to 25, which means a flow is determined to be non-congested at least after (please see Section VI-A.2) 25 LPS intervals equivalent to one contracting interval.

The parameter δ 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, β is set to 0.95, and $\triangle r$ is set to 0.0005. Also note that, in the experiments, packet drops are not allowed in any network node. This is because we would like to see performance of the schemes in terms of assured service.

A. Experiments on Single-bottleneck Topology

We run simulation experiments for PFCC and POCC on the single-bottleneck topology, which is represented in Figure 4-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.

For POCC, there is an additional component in the simulation: edge queues. The edge queues mark the packets when queue size exceeds 200 packets. So, in order to manage the edge queues in this simulation experiment, we simultaneously employ the two techniques defined in Section VI-B.3.

In terms of results, the volume given to each flow is very important. Figures 5-a and 6-a show the volumes given to each flow in PFCC and POCC respectively. Just to see a more smooth view, Figures 5-b and 6-b show the volumes averaged over 200 contract periods. We see the flows are sharing the bottleneck capacity almost in proportion to their budgets. The distortion in volume allocation is caused because of the fact that each flow is sharing the bottleneck capacity in proportion to their budgets when they are congested. When they are not congested, however, they return back to their individual capacity estimation. We can observer this dynamic in Figures 5-a and 6-a. In comparison to POCC, PFCC allocates volume more smoothly but with the same proportionality to the flows. The noisy volume allocation in POCC is caused by coordination issues (i.e. parameter mapping, edge queues) investigated in Section VI-B.2.

Figures 5-c and 6-c show the price being advertised to flows in PFCC and POCC respectively. As the new users join in, the pricing schemes increase 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.

Figures 5-d and 6-d shows the bottleneck queue size in PFCC and POCC respectively. 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. In comparison to PFCC, POCC manages the bottleneck queue much better because of the tight control enforced by the underlying edge-to-edge congestion control algorithm Riviera.

Figures from 7-a to 7-c show the sizes of edge queues in POCC. We can observe that users get active at 5000s of intervals. We observe stable behavior but with oscillations larger than the bottleneck queue illustrated in Figure 6-d. This is because of the tight edge-to-edge congestion control, which pushes backlog to the edges. Also, observe that the edge queues are generally much lower than the threshold of 200 packets. This means that the packets were marked at the edge queues very rarely. So, the technique of marking the packets at the edges and reducing the estimated capacity indirectly was not dominant in this simulation. Rather, the technique of reducing the estimated capacity directly at the ingress was dominant in terms of handling of edge queues (please see Section VI-B.3 for full understanding of these two techniques).

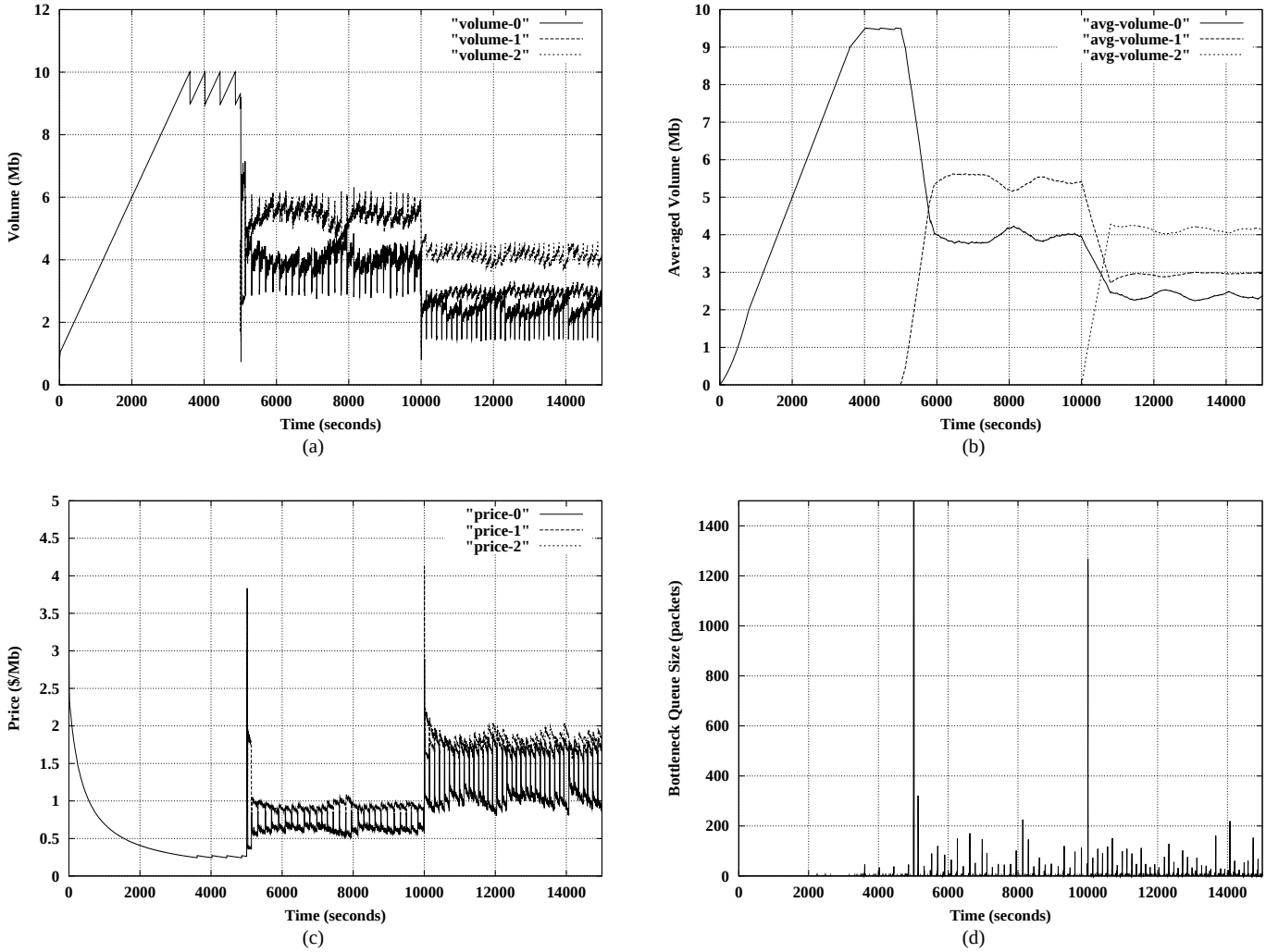


Fig. 5. Results of single-bottleneck experiment for PFCC: (a) Volumes contracted by each flow. (b) Average of contracted volumes over 200 contracts. (c) Price advertised to the flows. (d) Bottleneck queue length.

In PFCC, average utilization of the bottleneck link was more than 90%, and no packet drops were allowed. In POCC, average utilization of the bottleneck link was 80%. This is because of the fact that Riviera's parameters are set such that it provides 80% utilization of the network. So, as we claimed in Section VI-B.2, the limitations of the underlying congestion control scheme also limits the overall performance. We specifically set the parameters of Riviera to show this overall behavior, in reality Riviera can provide higher utilization given that the parameters are set properly.

B. Experiments on Multi-bottleneck Topology

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

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

Since Riviera does not currently [5] provide a set of parameters for weighted allocation on multi-bottleneck topology, we will not run any experiment for POCC on multi-bottleneck topology.

In order to illustrate Property 1, we run a series of experiments for PFCC with different α values. Remember that α is the fairness coefficient of Distributed-DCC. Higher α values imply more penalty to the flows that cause more congestion costs. We use a larger version of the topology represented in Figure 4-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. All the other users have traffic flows crossing the long flow. 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. Notice that the number of bottlenecks in the system is one less than the number of active user flows. We are interested in the volume given to the long flow, since it is the one that cause more congestion costs than the other user flows.

Figure 8-a shows the average volume given to the long flow versus the number of bottlenecks in the system for different val-

[5] It is still being studied by its developers.

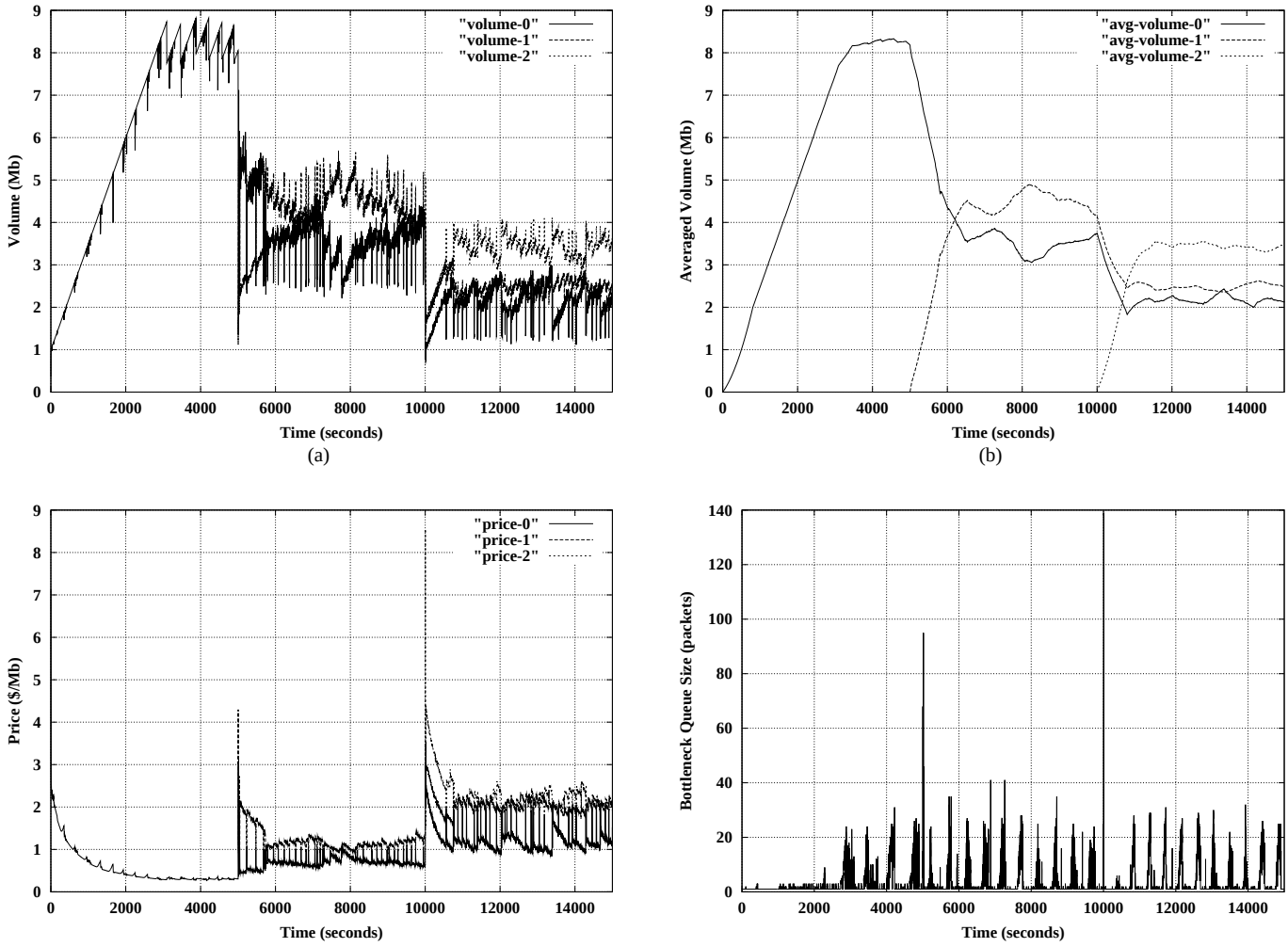


Fig. 6. Results of single-bottleneck experiment for POCC: (a) Volumes contracted by each flow. (b) Average of contracted volumes over 200 contracts. (c) Price advertised to the flows. (d) Bottleneck queue length.

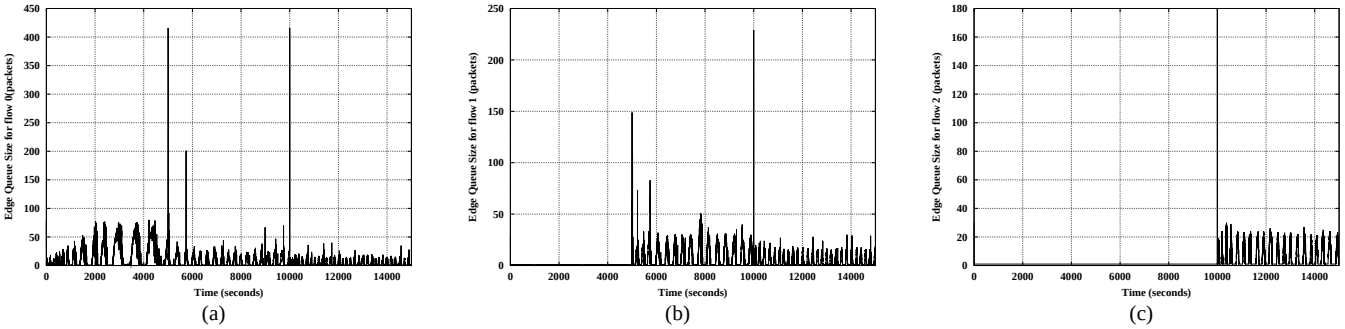


Fig. 7. Sizes of edge queues in the single-bottleneck experiment for POCC: (a) Edge queue for flow 0. (b) Edge queue for flow 1. (c) Edge queue for flow 2.

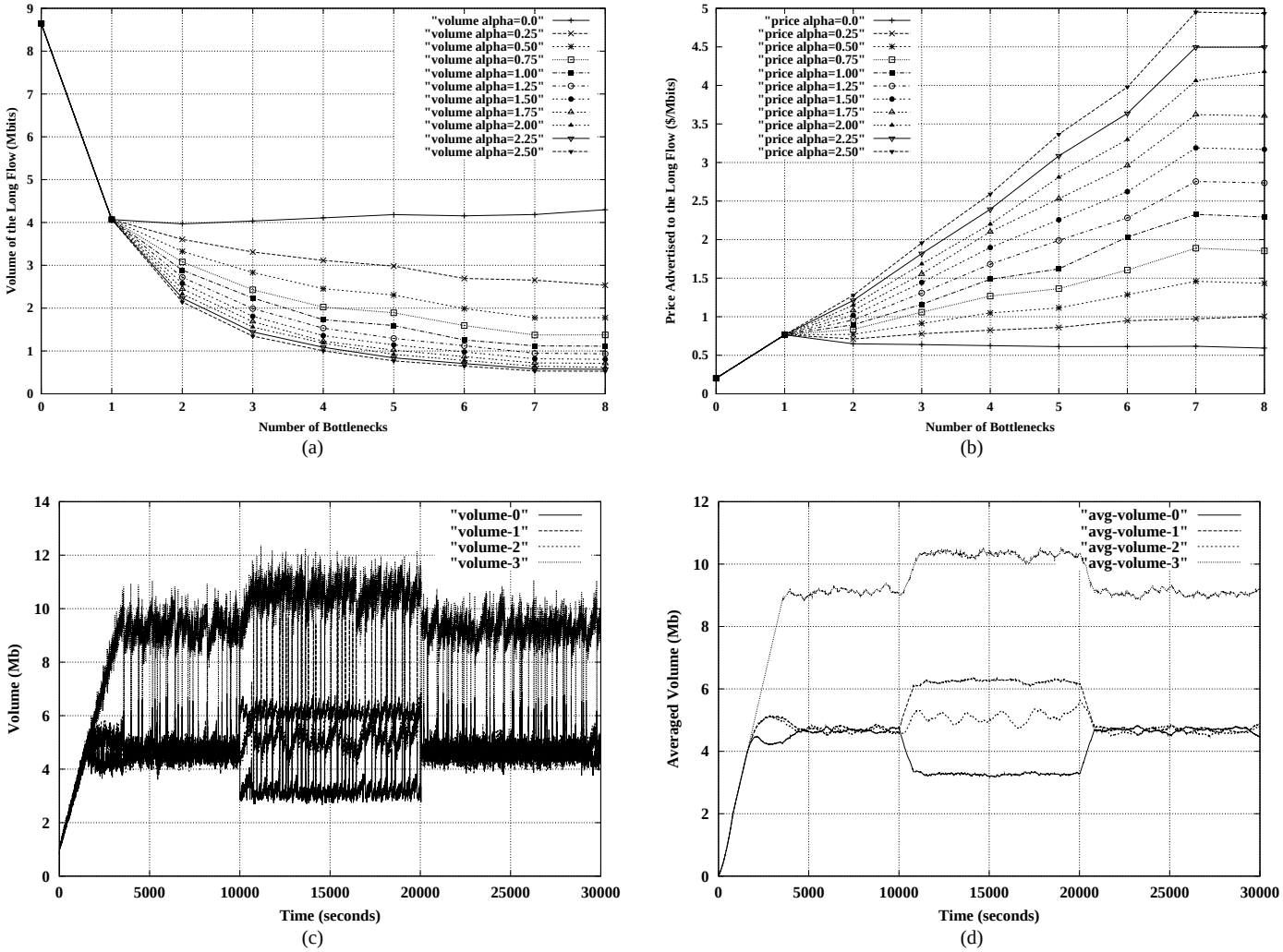


Fig. 8. Results of PFCC experiments on multi-bottleneck topology: (a) Volume given to the long flow in a linear network. (b) Price advertised to the long flow in a linear network. (c) Volumes given to the flows. (d) Volumes given to the flows averaged for 200 contracts.

ues of α. As expected the long flow gets less and less capacity as α increases. When α is zero, the scheme achieves max-min fairness. As it increases the scheme gets closer to proportional fairness. Also note that, the other user flows get the rest of the bottleneck capacity, and hence utilize the bottlenecks.

This variation in fairness is basically achieved by advertisement of different prices to the user flows according to the costs incurred by them. Figure 8-b 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.

Finally, to illustrate Property 2, we ran an experiment on the topology in Figure 4-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 4-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 (e.g. marking thresholds, initial values) are exactly the same as in the single-bottleneck experiments of the previous section.

Figure 8-c shows the volumes given to each flow, and Figure 8-d 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 individual flows by the capacity allocation algorithm (see Section VI-A.2).

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 and Discussions

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*

overlay 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.

Based on the way of approaching to the congestion control problem, we developed two pricing schemes (PFCC and POCC) within the Distributed-DCC framework. As a novel approach, distinguishing feature of POCC is to overlay pricing on top of edge-to-edge congestion control. By comparative evaluation of PFCC and POCC, we showed that POCC performs better in terms of managing congestion in network core because of the tight (low time-scale) control enforced by the underlying edge-to-edge congestion control mechanism. However, we also showed that overall performance (e.g. fairness, utilization) is dependent on the flexibility of the underlying edge-to-edge congestion control mechanism.

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), better estimation of the parameter r_{ij} (see (5)).

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 IWQoS*, 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, "The economics of the Internet: Utility, utilization, pricing, and quality of service," Tech. Rep., AT & T Labs, 1998.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] A. C. Chiang, *Fundamental Methods of Mathematical Economics*, McGraw Hill, Inc., 1984.

[23] D. M. Chiu, "Some observations on fairness of bandwidth sharing," Tech. Rep. TR-99-80, Sun Microsystems Labs, September 1999.

[24] J. K. MacKie-Mason and H. R. Varian, "Pricing the congestible network resources," *IEEE Journal on Selected Areas of Communications*, vol. 13, pp. 1141–1149, 1995.

[25] D. Harrison, S. Kalyanaraman, and S. Ramakrishnan, "Overlay bandwidth services: Basic framework and edge-to-edge closed-loop building block," Poster in SIGCOMM, 2001.

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